October 14, 1999

SmartLCDÔ

Programmer's Guide
And
Application Notes

The following documentation is supporting material for the SmartLCD™ sample programs. Refer to the SmartLCD™ sample programs mentioned in this document.

1. Video Memory and Frames

1.1 OVERVIEW

The SmartLCD can display layered text and graphics and scroll the display in any direction.

1.2 FRAMES

A **frame** selects the portion of memory to be displayed on the SmartLCD. A **graphics frame** encompasses a 240 row by 40-byte area of memory. Each byte within the graphics frame is displayed as an 8-pixel horizontal segment (Figure 1.0). A **text frame** encompasses a 30 row by 40-byte area of memory. Each byte within the text frame is displayed as an 8-pixel by 8-pixel ASCII character.

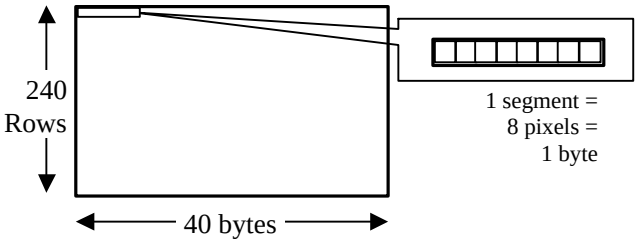


Fig. 1.0 Frame Layout

1.3 Multiple Frames

The SmartLCD supports two types of multi-frame configurations.

- **Dual Frame Configuration (default)**

  Figure 1.1 shows the default dual frame configuration. The dual frame configuration supports one text frame and one graphics frame.

  Frame 1 (Text): Frame 1 in figure 1.1 starts at address 0x0000 in the video memory and ends at 0x04AF (1200 bytes). Each byte in frame 1 is, by default, displayed on the SmartLCD as an 8x8 pixel letter. 1200 letters can be displayed on the LCD at one time (30 rows of 40 8x8 pixel letters).

  Frame 2 (Graphics): Frame 2 in figure 1.1 starts at address 0x1000 in the video memory and ends at 0x357F (9600 bytes). Each byte in frame 2 is displayed as an 8x1 horizontal pixel image. Frame 2 displays 9600 bytes: 240 rows of 40 bytes.

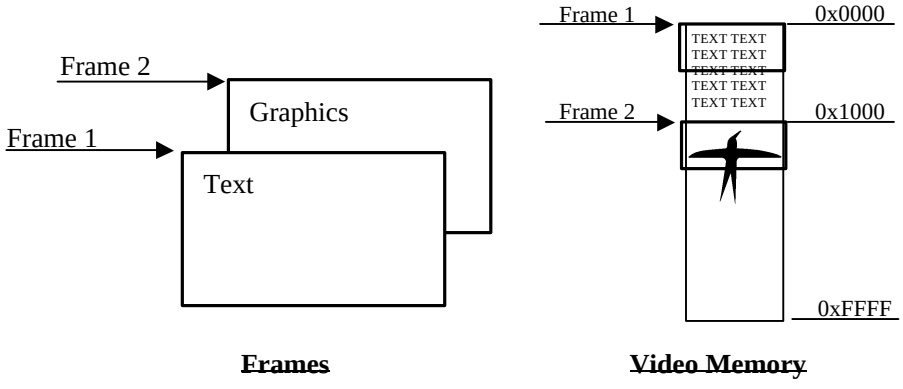


Fig. 1.1 Dual Frame Configuration (default)

- **Triple Frame Configuration**

  Figure 1.2 shows the triple frame configuration. The triple frame configuration supports graphic frames only.

  Frame 1, 2, and 3 (Graphics): Frame 1 starts at address 0x0000, frame 2 starts at address 0x6000 and frame 3 starts at address 0xC000. Each byte in a frame is displayed as an 8x1 horizontal pixel image. Each frame displays 9600 bytes: 240 rows of 40 bytes.

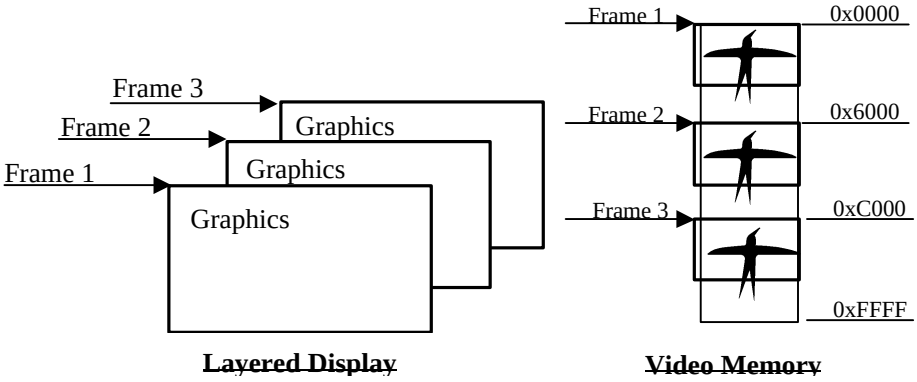


Fig. 1.2 Triple Frame Configuration (default)

1.4 OVERLAY COMPOSITION

The SmartLCD composes the LCD display by OR'ing, AND'ing, or Exclusive-OR'ing the overlapping pixels of each frame (Figure 1.3).

Overlay	Composition	Function
0	OR	Frame 1 ***OR*** Frame 2 ***OR*** Frame 3
1	Exclusive-OR	(Frame 1 ***XOR*** Frame 2) ***OR*** Frame 3
2	AND	(Frame 1 ***AND*** Frame 2) ***OR*** Frame 3

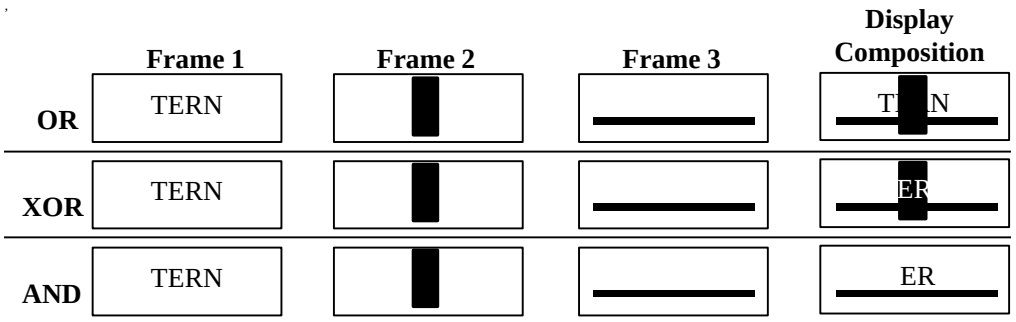


Fig. 1.3 Overlay Composition

1.5 Virtual Screen

The area of memory that is not encompassed by a frame is considered the virtual screen. The data in the virtual screen can be displayed by changing the starting address of a frame. The virtual screen makes vertical and horizontal scrolling possible.

- **Vertical Scrolling**
  By changing a frame's start address plus or minus by one horizontal line (40 bytes by default) the LCD display will smoothly scroll up or down through the video memory. Figure 1.4 shows the effects of this process.

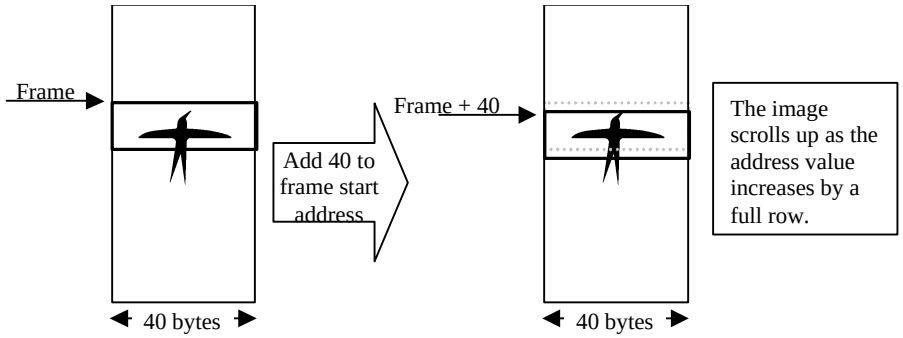


Fig. 1.4 Vertical Address Range

- **Horizontal Scrolling**
  By default, the horizontal address range of the virtual screen is 40 bytes (320 pixels). The width of the SmartLCD display is also 40 bytes (320 pixels). Thus, moving a frame's start address by 39 bytes or less will cause the frame to overlap the image (figure 1.5a).

  The SmartLCD allows the user to expand the horizontal memory address range of the virtual screen to provide smooth horizontal scrolling (Figure 1.5b).

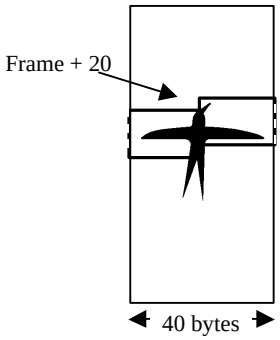


Fig. 1.5a Overlapped Frame

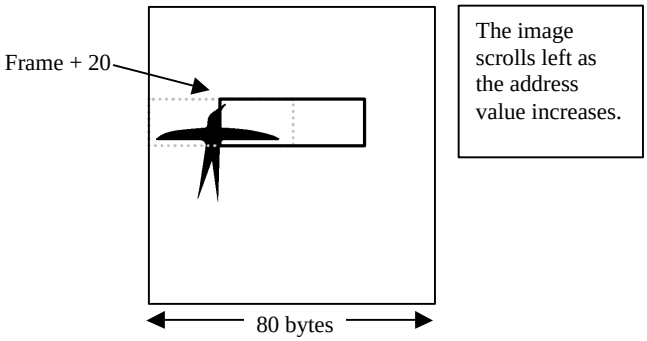


Fig. 1.5b Expanded Horizontal Memory

2. Smart LCD Libraries (SLB)

The TERN SL40 sample programs disk includes a number of structures and functions that simplify the programming process of the SL40. The following table lists the files that contain these structures and functions.

- **Object and Header Files**

Object File	Header File	Description
slb_lib.c	slb_lib.h	SL structure and screen setup/manipulation routines.
slb_pict.c	slb_pict.h	Picture structure and bitmapped picture display routines.
slb_bttn.c	slb_bttn.h	Push button structure and routines.
slb_fill.c	slb_fill.h	Fill bar structure and routines.
slb_grph.c	slb_grph.h	Graph structure and functions.
slb_font.c	slb_font.h	Custom user font routines.

2.1 SLB_LIB

Slb_lib defines the SL structure and substructures that configure the SmartLCD.

- **SL (main structure)**

Type	Member	Description
`Display`	*display*	Display structure.
`Cursor`	*cursor*	Cursor structure.
`Frame`	*frame1*	Frame structure.
`Frame`	*frame2*	Frame structure.
`Frame`	*frame3*	Frame structure.
`Font`	*font*	Font structure.
`unsigned char`	*lock*	Denotes graphic controller chip in use.

- **Display**

Type	Member	Description
`unsigned char`	*on*	Turn LCD on = 1, Off = 0.
`unsigned int`	*horizontal_memory_range*	Horizontal address range of the virtual screen.
`unsigned char`	*num_frames*	2 (txt,graph) or 3 (graph,graph,graph) Frames.
`unsigned char`	*overlay*	Frame overlay 0=OR, 1=XOR, 2=AND.

- **Cursor**

Type	Member	Description
`unsigned char`	*state*	Cursor state: 0 = off, 1 = on, 2,3 = flashing.
`unsigned char`	*direction*	Cursor direction: 0=forward,1=back,2=up,3=down.
`unsigned char`	*height*	Cursor height: val. 1-15 = height 2-16 lines.
`unsigned char`	*width*	Cursor width: val. 0-15 = width 1-16 pixels.

- **Frame**

Type	Member	Description
`unsigned char`	*state*	Frame state: 0 = off, 1 = on, 2,3 = flashing.
`unsigned int`	*mem_start_address*	Start address of a frame.

- **Font**

Type	Member	Description
`unsigned char`	*mode*	Font mode: default = 0, user's custom font = 1.
`unsigned int`	*mem_start_address*	Start address of font image.
`unsigned char`	*width*	Character pixel width: 8 or 16 pixels.

`unsigned char`	*height*	Character pixel height: 8 or 16 pixels.

- **sl_cmd_wr(unsigned char cmd)**
  Writes a byte to the SmartLCD command register.

- **sl_dat _wr(unsigned char data)**
  Writes a byte to the SmartLCD data register.

- **sl_dat_rd()**
  Read a byte from the SmartLCD data register.

- **sl_flag_rd()**
  Read a byte from the SmartLCD status flag register.

- **sl_struct_init(SL *)**
  This function initializes the SL structure. The default settings are listed below.

SL Member	Init. Value	Description
display.on	1	Display on
display.horizontal_memory_range	40	Horizontal memory = 40 bytes
display.num_frames	2	2 frames (text, graphics)
display.overlay	1	Xor overlay
cursor.state	0	Cursor off
cursor.direction	0	Advance forward
cursor.height	2	Height = 2 rows
cursor.width	8	Width = 8 pixels
frame1.state	1	Frame 1 on
frame1.mem_start_address	FRAME1	Macro defined in slb_lib.h
frame2.state	1	Frame 2 on
frame2.mem_start_address	FRAME2	Macro defined in slb_lib.h
frame3.state	0	Frame 3 off
frame3.mem_start_address	FRAME3	Macro defined in slb_lib.h
font.mode	0	Default font
font.mem_start_address	FONT_ADDRESS	Macro defined in slb_lib.h
font.width	8	8-pixel width
font.height	8	8-pixel height
lock	0	Unlocked

- **sl_set_mode(SL *)**
  Invokes all of the `sl_set` functions.

- **sl_set_system(SL *)**
  Sets the following attributes:

```
SL.font.mode
SL.font.width
SL.font.height
SL.display.horizontal_memory_range
```

- **sl_set_frames (SL *)**
  Sets the following attributes:
  ```
  SL.frame1.mem_start_address
  SL.frame2.mem_start_address
  SL.frame3.mem_start_address
  ```

- **sl_set_overlay(SL *)**
  Sets the following attributes:
  ```
  SL.display.overlay
  ```

- **sl_set_cursor (SL *)**
  Sets the following attributes:
  ```
  SL.cursor.width
  SL.cursor.height
  SL.cursor.direction
  ```

- **sl_set_display (SL *)**
  Sets the following attributes:
  ```
  SL.frame1.state
  SL.frame2.state
  SL.frame3.state
  SL.cursor.state
  ```

- **sl_put_cursor (SL *, unsigned int cursor)**
  Sets the cursor to location *cursor* in memory.

- **sl_putstr (SL *, unsigned int addr, unsigned char far * text)**
  Stores C formatted string into video memory starting at *address*.

- **sl_clear_all (SL *)**
  Sets every byte in video memory to 0x00

- **sl_clear_frame (SL *, char frame)**
  Sets every byte in a frame (1, 2, 3) to 0x00.

2.2 SLB_PICT Library

SLB_PICT contains functions and structures that display bitmapped images created by `BMP2C.EXE` (see section 4).

- **Picutre**

Type	Member	Description
`int`	*width*	Picture width in pixels.
`int`	*height*	Picture height in pixels.
`int`	*numBytes*	Number of whole bytes in the image.
`unsigned char far *`	*image*	Pointer to image array.

- **void sl_put_picture(SL *, unsigned int address, Picture *)**
  Store *picture* in video memory starting at *address.*

2.3 SLB_GPX Library

SLB_GPX contains functions that create graphic objects.

- **void sl_draw_pixel(SL *, unsigned int orig_addr, int x, int y)**
- **void sl_erase_pixel(SL *, unsigned int orig_addr, int x, int y)**
  Draw/erase a single pixel at location (x,y). Orig_addr is the address in video memory of (0,0).

- **void sl_draw_line(SL *, unsigned int orig_addr, int x1, int y1, int x2, int y2)**
- **void sl_erase_line(SL *, unsigned int orig_addr, int x1, int y1, int x2, int y2)**
  Draw/erase a line from (x1, y1) to (x2, y2).

- **void sl_draw_box(SL *, unsigned int orig_addr, int x1, int y1, int x2, int y2)**
- **void sl_erase_box(SL *, unsigned int orig_addr, int x1, int y1, int x2, int y2)**
  Draw/erase an empty box with a corner-to-corner dimension of (x1, y1) to (x2, y2).

- **void sl_draw_fill_box(SL *, unsigned int orig_addr, int x1, int y1, int x2, int y2)**
- **void sl_erase_fill_box(SL *, unsigned int orig_addr, int x1, int y1, int x2, int y2)**
  Draw/erase a solid box with a corner-to-corner dimension of (x1, y1) to (x2, y2).

- **void sl_draw_circle (SL *, unsigned int orig_addr, int x1, int y1, int r)**
- **void sl_erase_circle(SL *, unsigned int orig_addr, int x1, int y1, int r)**
  Draw/erase a circle with a center at (x1, y1) and a radius r.

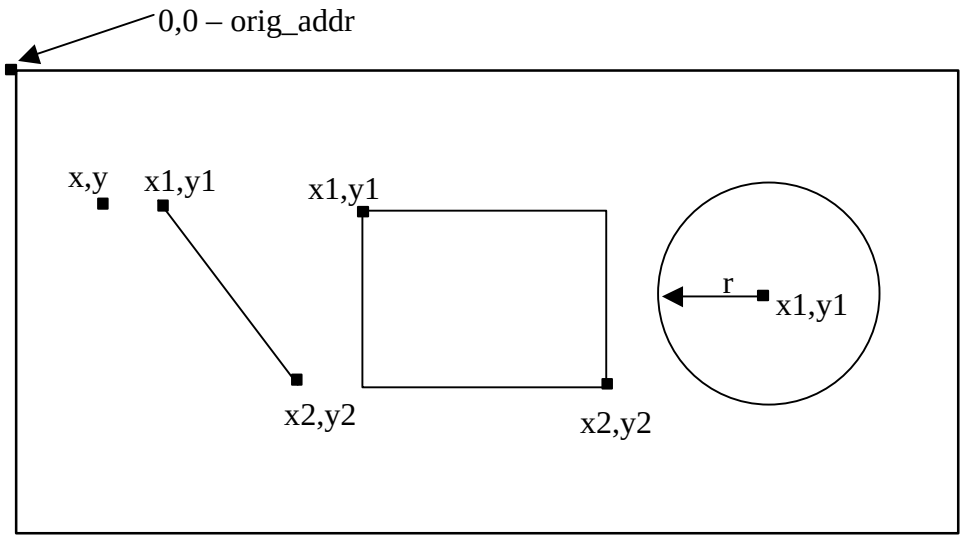


Fig. 5.1 SLB_GPX Graphics

note:

- Floating point emulator must be enabled in the makefile to use graphs.

2.3 SLB_BTTN Library

SLB_BTTN contains functions and structures that create and control button objects.

- **Button**

Type	Member	Description
`unsigned int`	frame	Frame address used to display button.
`unsigned char`	r1	Top left corner touch pad row 1..7
`unsigned char`	c1	Top left corner touch pad column 1..10
`unsigned char`	r2	Bottom right corner touch pad row 1..7
`unsigned char`	c2	Bottom right corner touch pad column 1..10
`unsigned char`	state	Button pressed=1, not pressed=0
`char`	name[30]	Button Label

- **void put_button(SL *, button *)**
  Display button and button label

- **int push_button(SL *, button *)**
  Checks if a button is depressed or not, returns 1 if the button is depressed, else returns 0.

- **void clear_button(SL *, button *)**
  Clears a button and button label.

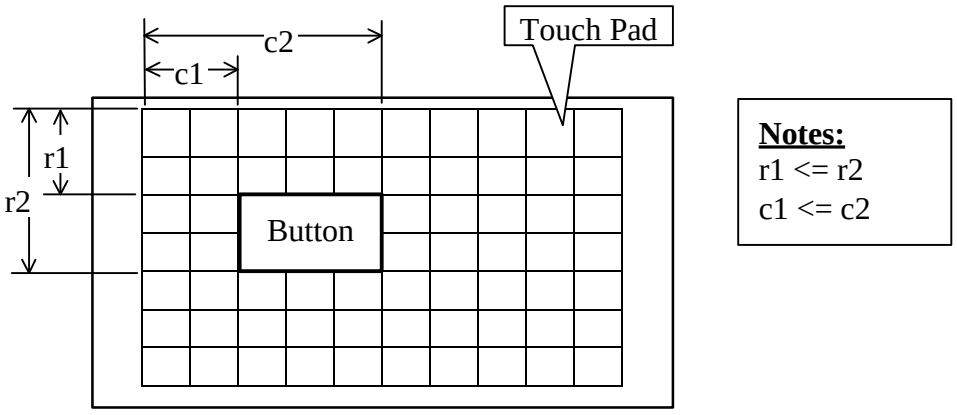


Fig. 5.1 SL40 Button Layout

2.4 SLB_FILL Library

SLB_FILL contains functions and structures that create and control fill bars. A fill bar displays a numeric value graphically using a scaled bar. Use the fill bar to display a numeric value between a minimum and maximum value such as an ADC reading.

- **FillBar**

Type	Member	Description
`unsigned int`	*frame*	Frame address used to display fill bar.
`unsigned char`	*top*	Top placement of the fill bar 1..240, 1 unit = 1 pixel
`unsigned char`	*bottom*	Bottom placement of the fill bar 1..240, 1 unit = 1 pixels
`unsigned char`	*width*	Width of fill bar 1..40, 1 unit = 8 pixels
`unsigned char`	*pos*	Position of the fill bar 1..40, 1 unit = 8 pixels
`double`	*minval*	Minimum input value
`double`	*maxval*	Maximum input value
`double`	*val*	Current input value

- **void put_bar(SL *, fillBar *)**
  Display a fill bar. Use **put_bar()** to display a new fill bar or to update **val** of an existing fill bar.

Notes:

- Floating point emulator must be enabled in the makefile to use bar graphs.
- Set top and bottom far away from each other for better resolution.
- If the fill bar is updated frequently, set the width as small as possible to prevent a strobe effect.

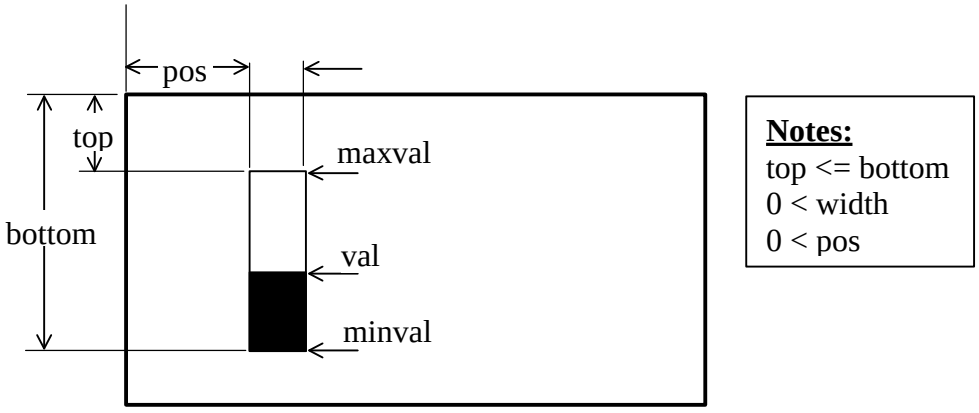


Fig. 6.1 SL40 Fill Bar Layout

2.5 SLB_GRPH Library

SLB_GRPH contains functions and structures that plot graphs. A graph displays an (x,y) coordinate on a two-dimensional Cartesian plot.

- **Graph**

Type	Member	Description
`unsigned int`	*frame*	Frame address used to display graph.
`unsigned char`	*top*	Top placement of the graph 1..240, 1 unit = 1 pixel
`unsigned char`	*bottom*	Bottom placement of the graph 1..240, 1 unit = 1 pixel
`unsigned char`	*width*	Width of the graph 1..40, 1 unit = 8 pixels
`unsigned char`	*pos*	Position of the graph 1..40, 1 unit = 8 pixels
`double`	*minX*	Minimum X input value
`double`	*minY*	Minimum Y input value
`double`	*maxX*	Maximum X input value
`double`	*maxY*	Maximum Y input value
`double`	*pixX*	X value of a pixel. DO NOT MODIFY.
`double`	*pixY*	Y value of a pixel. DO NOT MODIFY.
`double`	*X*	x-coordinate of a point
`double`	*Y*	y-coordinate of a point

- **void graph_init(SL *, graph *)**
  Display a graph border. Use to initialize a new graph or to clear the data from an existing graph.
- **void plot_graph(SL *, graph *)**
  Plot the (X,Y) coordinate of the graph object.

notes:

- Floating point emulator must be enabled in the makefile to use graphs.
- To plot continuous points along the x-axis:
  1. Set minX = 0 and maxX = width*8. Each pixel width will equal 1.
  2. Loop from 0 to maxX by 1. Set the Y value for each value of X.

```
for( i = 0;  i < maxX; i++){
        graph.X = i;
        graph.Y = value;
        plot_graph(&graph);
}
```

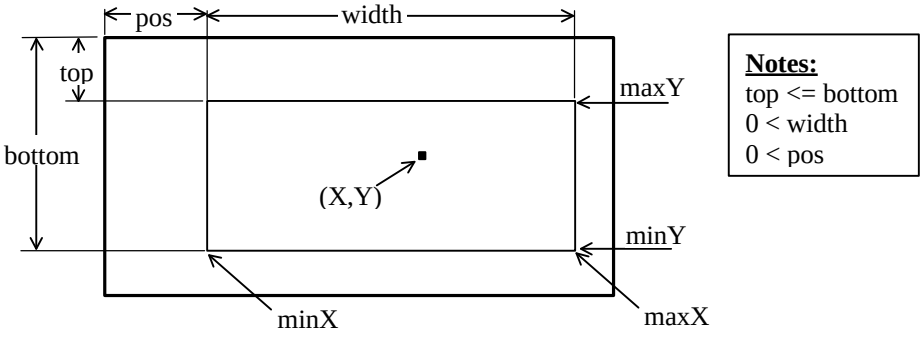


Fig. 7.1 SL40 Graph Layout

3. Additional Software

3.1 bmp2c.exe

Bmp2c.exe is a DOS executable program that converts a bit mapped file (BMP) into a "C" style character array (figure 4.1). The character array can be used to display an image on the SL40.

3.1.1 *Generating a Character Array from a BMP File*

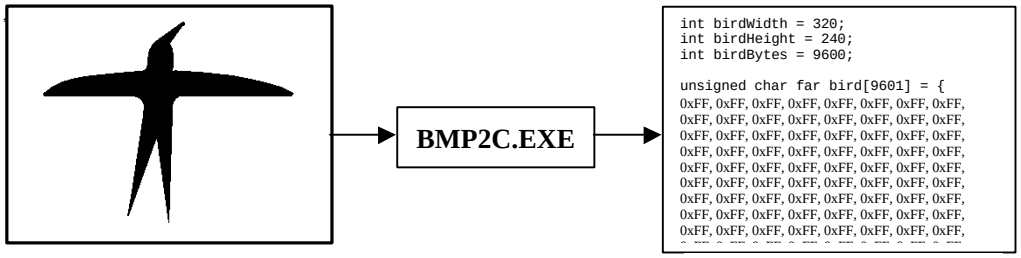
- Design a black and white image using MSPaint or another graphics utility. Images must be black and white, less than or equal to 320x240 pixels and saved as .BMP file.
- Use bmp2c.exe to convert your bitmapped image to a C header file.
  Syntax: `bmp2c file.bmp`
  Returns: `file.h`

3.1.2 *Contents of the C Header File*

- `int fileWidth` : Pixel width of image
- `int fileHeight` : Pixel height of image
- `int fileBytes` : Total number of bytes in the image
- `unsigned char far` *`file`*`[]` : The far data for the bitmapped image

3.1.3 *Displaying an Image*

- See **SL_IMAGE.C** for example
- Include "file.h" into main source file:
  `#include "file.h"`
- Pass the image pointer to **`put_picture(SL *, unsigned int addr, Pict *)`** to display the image. Set ***address*** to the video memory start address of the picture (See section 2 and 3).



bird.bmp **bird.h**

Fig. BMP2C.EXE

3.2 txt2c.exe

Txt2c.exe is a DOS executable program that converts a text file (BMP) into a formatted "C" style character array. The character array can be used to display text on the SL40.

3.2.1 *Generating a Character Array from a TXT File*

- Create a text (.txt) file.
- Use txt2c.exe to convert your text file into a C header file.
  Syntax: `txt2c file1.txt [file2]`
  Returns: `file.h`

3.2.2 *Displaying an Image*

- See **SL_TXT.C** for example
- Include "file.h" into main source file:
  ```
  #include "file.h"
  ```
- Use **`put_str(SL*, unsigned int addr, unsignedc char far* text)`** to display the image.