MF1383-01

EPSON

CMOS 32-BIT SINGLE CHIP MICROCOMPUTER **E0C33 Family**

GRAPHIC33 MIDDLEWARE MANUAL

ENERGY SAVING EPSON

SEIKO EPSON CORPORATION

PREFACE

This manual describes the configution and functions of CompactFlash Middleware GRAPHIC33 for the E0C33 Family, and explains methods for using this middleware. It is targeted to developers of applications for the E0C33 Family of microcomputers.

CONTENTS

1 Outline of GRAPHIC 33 Middleware

The GRAPHIC33 is a graphic middleware for the E0C33 Family of microcomputers. It is designed to perform the 2D graphics and GUI processing on the E0C33 Family chip. Graphics functions are supplied as library functions, which can be used by linking with the target program.
The product also contains PC software tools for creating Image ROM data.
GRAPHIC33 middleware is ideally suited to the development of applications such as cellular phone, PDAs, electronic stationery, and toys.

The main features of the GRAPHIC33 middleware are given below:

- Support for EOC33 Family which comes with built-in LCDcontroller
- Drawing primitive functions

  Drawing functions such as points, straight lines, circles, ellipses, rectangles, arcs, or fill.
- Functions to display and save picture images
- Functions to display text
- Graphical user interface functions (To use the GUI function, you need "ROS33")
- Tools for making image data as ROM data

CAUTION

- Be sure to fully evaluate the operation of your application system before shipping. Seiko Epson assumes no responsibility for problems arising from use of this middleware in your commercial products.
- Rights to sell this middleware are owned solely by Seiko Epson. Resale rights are not transferred to any third party.
- All program files included in this package, except sample programs, are copyrighted by Seiko Epson. These files may not be reproduced, distributed, modified, or reverse-engineered without the written consent of Seiko Epson.

1.1 Components of the GRAPHIC 33 Package

The contents of the GRAPHIC33 package are listed below. When unpacking, check to see that all of the following items are included.

(1) Tool disk (CD-ROM)	1 pc.
(2) E0C33 Family GRAPHIC33 Middleware Manual (this manual)	1 pc. each in English and Japanese
(3) Warranty card	1 pc. each in English and Japanese

1.2 Basic Configuration of GRAPHIC33 System

Hardware configuration

The basic hardware structure of the GRAPHIC33 system including image input/output consists of the EOC33chip, external memory, and DMT33LCD as shown in "Figure 1.2.1".
Also, the GRAPHIC library uses 1 channel of the 16 bits programmable timer on the E0C33 chip.

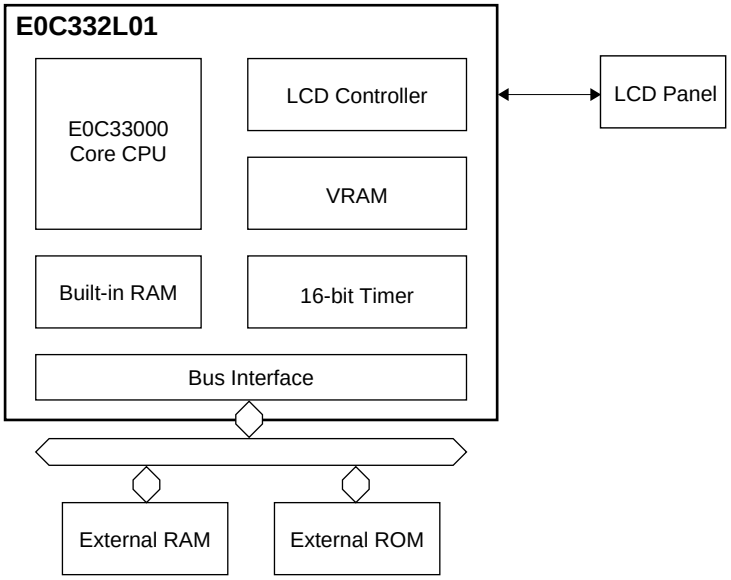


Figure 1.2.1 Hardware structure of the GRAPHIC33 system

Software configuration

The GRAPHIC33 library is a middleware, positioned between the E0C33 hardware and user applications, performing hardware control associated with primitive image drawing, image display, and text display. By including or linking the top-level functions supplied as C source files into user applications, you can easily perform graphics processing, without having to call GRAPHIC33 library functions directly from the applications.

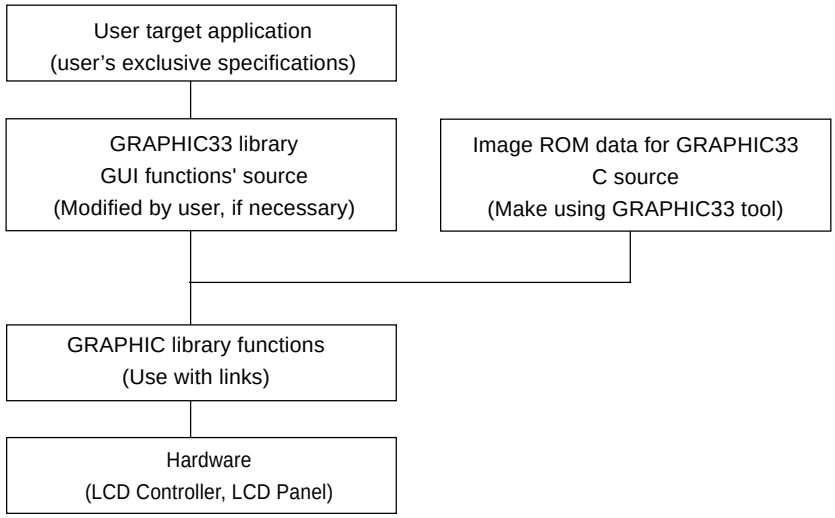


Figure 1.2.2 Software Configuration of Graphic33

For more details on GRAPHIC33 library functions and top-level functions, see Section 5, "GRAPHIC33 Library Reference".

GRAPHIC33 tools are PC software used to create and evaluate image ROM data and color palette. When complete, this data is downloaded to an E0C33 Family chip. All tools are 32-bit applications executable from a DOS prompt, and will run under Windows 95, Windows NT 4.0, or later Windows versions.
For detailed information on GRAPHIC33 tools, see Section 4, "GRAPHIC33 Tool Reference".

2 Installation

This section describes the operating environment for the GRAPHIC33 tools and explains how to install the GRAPHIC33 middleware.

2.1 Developing Environment

The following are the minimum requirements for developing software with the GRAPHIC33

Personal computer

An IBM PC/AT or fully-compatible machine. We recommend a Pentium 90 MHz or higher CPU, and 32 MB or more of RAM. A CD-ROM is required to install tools from the CD-ROM.

Display

An SVGA (800 × 600) monitor or better. In the Windows Control Panel, choose "Small font" among the display options.

System software

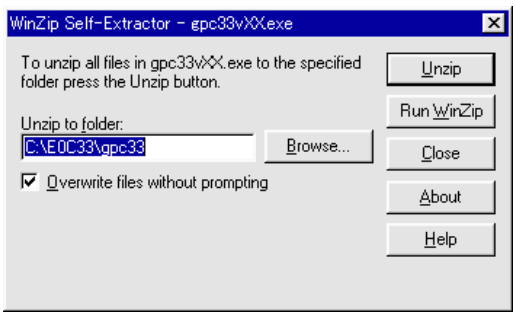
GRAPHIC33 tools run under Microsoft® Windows® 95 or Windows NT® 4.0, or later versions of Windows (English and Japanese versions).

Other

The "E0C33 Family C Compiler Package" is required for software development.
To use the GUI functions, you need "ROS33"

2.2 Method of Installation

The GRAPHIC33 library is supplied on CD-ROM. Open the self-extracting file on the CD-ROM named "gpc33vXX.exe" to install the GRAPHIC33 library in your computer. (The XX in this file name denotes a version number. For Version 1.0, for example, the file is named "gpc33v10.exe".)
Double-click on "gpc33vXX.exe" to start installation. The dialog box shown below appears.



Enter the path and folder name under which you want to install the files in the text box and click on the [Unzip] button. The specified folder is created and all files are copied into it.
If the specified folder already exists in the specified path and [Overwrite Files Without Prompting] is checked (turned on), the files in the folder are overwritten without asking for your confirmation.

The following shows the directories and file configuration after the program files have been copied:

```
(root)\
    readme.txt       Supplementary explanation, etc. (in English)
    readmeja.txt     Supplementary explanation, etc. (in Japanese)
    gpctool\         .....GRAPHIC33 tool directory
        readmeja.txt   Japanese manual
        readme.tx      English manual
        bin\           ....Tool executable file directory
                gpcgetbmp.exe   Bitmap file → Bitmap image data conversion tool
                gpcsel.exe      24-bit bitmap → 8-bit bitmap image data conversion tool
                gpcpldat.exe    Color palette file → C source conversion tool for color palette
                bin2c.exe       Binary → C source conversion tool for CC33
                bin2s.exe       Binary → Assembler source conversion tool
                bmp2rgb.exe     BMP → RGB format conversion tool
        sample\    .....Sample directory
                Image ROM data such as sample images, color palette, cursor, and mouse

    gpclib\          .....GRAPHIC33 library related items
        readmeja.txt   Japanese manual
        readme.txt     English manual
        lib\           .....GRAPHIC33 library directory
                gpc33.lib       GRAPHIC33 library
                ros33.lib       ROS33 library
                gpcpal.o        Palette library
        include\   .....Header file directory for GRAPHIC33 library functions
                gpc.h           Header file of graphics library
                gpcdrv.h        Header file of display driver
                gpcgui.h        Header file for GUI functions
                c33208.h        Header file for C33208 symbol definition
                itron.h         Header file of ROS33
                ros33.h         Header file of ROS33
                gpcpal.h        Header file for palette functions
        src\           .....GUI source directory
                gpcgui.c        GUI functions
                gpcevent.c      Event functions
                guiimage.c      GUI image data (cursor, mouse, UP button, DOWN button)
        libsrc\ .....Library source directory
                gpcdrv.c        Display driver
                gpcfont1.c      Graphics font (1 byte)
                gpcfont2.c      Graphics font (2 bytes)
                gpccolor.c      Color table
        sample\    .....Sample directory
                (For details on the sample programs 'structure and their use, refer to the "readme.txt" or
                "readmeja.txt" file in "gpclib".)
            gpc\           .....Graphics function sample directory
            gpc_k\         .....GUI key sample directory
            gpc_m\         .....GUI mouse sample directory
            gpcpal\        .....Palette sample directory
            gpcpal2\       .....Palette sample directory (No.2)
```

Although you can select a different directory structure and file organization, the discussions in the following pages will assume that the files have been copied from the CD-ROM according to the directory structure given above.

3 Software Development

This section describes how to develop Graphics processing software on the E0C33 Family chip. Shown below is the basic flow of development:

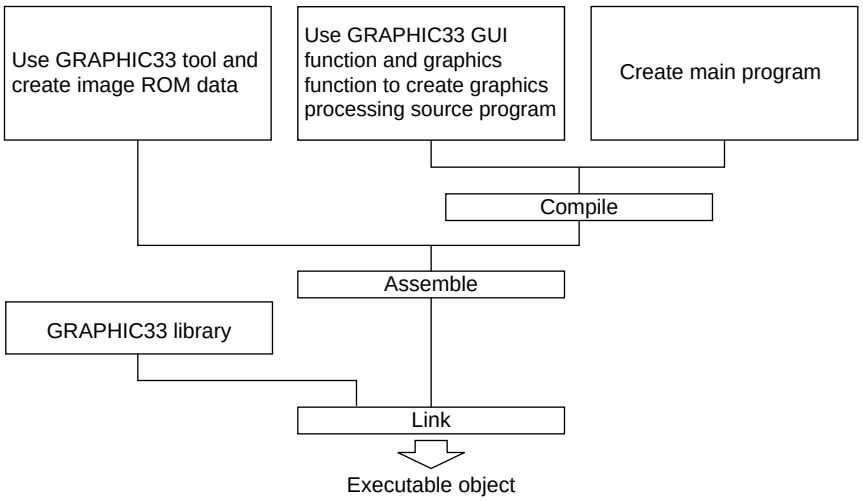


Figure 3.1 E0C33 graphics software development procedure

1) Using GRAPHIC33 tools, create the image ROM data from your image file, and convert into a C source file.

2) Create a user application. For graphics processing, use GUI function and graphics function provided in the GRAPHIC33 library. You can include the source file for the image ROM data created in Step1 in the user application source.

3) Compile and assemble the source program.

4) Link the objects generated in Step 3 with the GRAPHIC33 library, and then generate an executable object file.

3.1 Creating Image ROM Data by Using the GRAPHIC33 Tools

Figure 3.1.1 shows procedures of creating image ROM data with GRAPHIC33 tools.

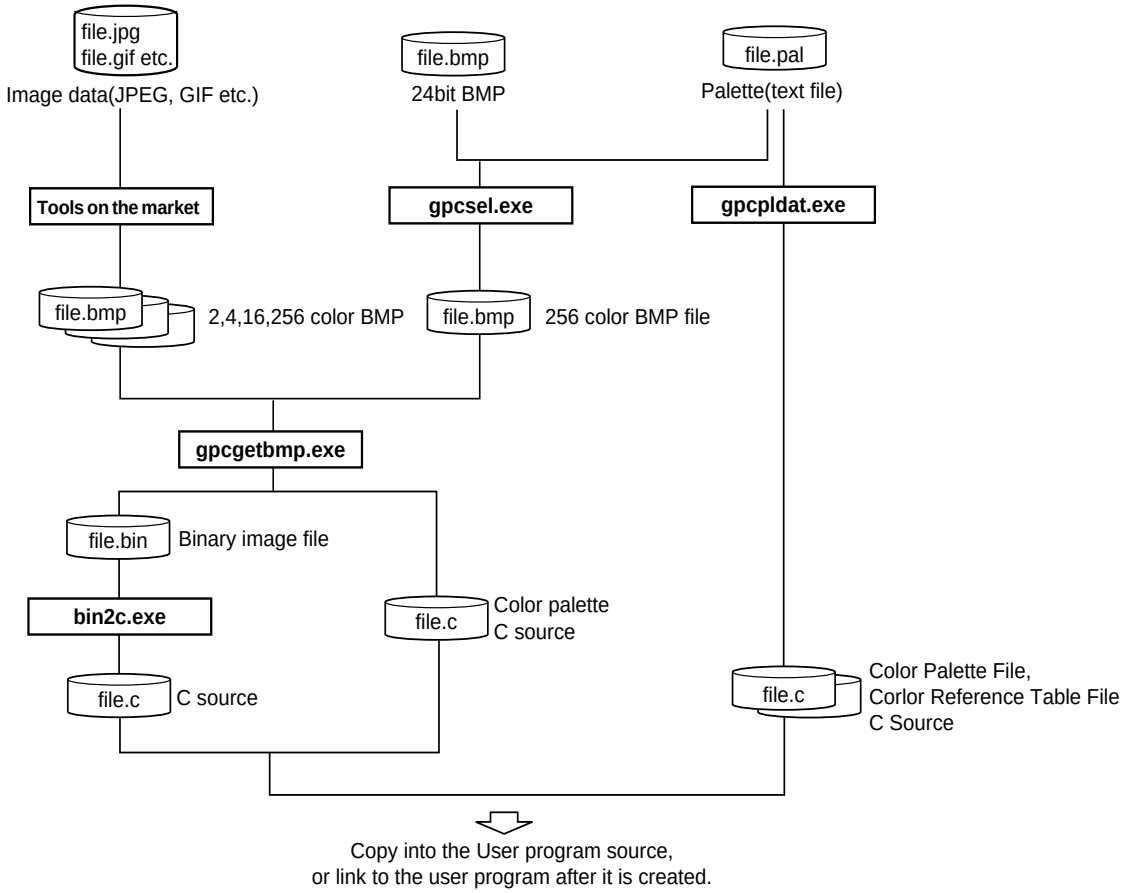


Figure 3.1.1 Flowchart for Creating Image ROM DATA

This provides an outline of usage of the GRAPHIC33 tools. For more details, see Section 4, "GRAPHIC33 Tool Reference".
The following explanation assumes that PATH is set to the "gpctool\bin\" directory.

Example: `DOS>PATH c:\e0c33\gpc33\gpctool\bin`

3.1.1 Outline of Image ROM DATA

Image Data Format (Binary Image File)

(The values are expressed as hexadecimal numbers delimited by a comma.)
Each number indicates the index value of the color of the color palette.
The bit number of each index value varies according to its BPP, as shown below.
For 1 BBP, 1 bit per pixel
For 2 BBP, 2 bits per pixel
For 4 BBP, 4 bits per pixel
For 8 BBP, 8 bits per pixel

The following files use 8 BPP as examples.

```
0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7,
0x82, 0x50, 0x25, 0x25, 0x25, 0x25, 0x25, 0x25,
0x24, 0x24, 0x24, 0x24, 0x24, 0x24, 0x24, 0x4e,
0x24, 0x4e, 0x4e, 0x4e, 0x72, 0x4e, 0x79, 0xab,
            .
            .
            .
0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7,
0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7
```

Color palette

- For the 16 colors or 256 colors, create the palette so that the following definitions can be used.
  The following 16 colors are reserved for Graphic33.

```
#define GPC_BLACK               0
#define GPC_RED                 1
#define GPC_GREEN               2
#define GPC_YELLOW              3
#define GPC_BLUE                4
#define GPC_MAGENTA             5
#define GPC_CYAN                6
#define GPC_LIGHTGRAY           7
#define GPC_DARKGRAY            8
#define GPC_LIGHTRED            9
#define GPC_LIGHTGREEN          10
#define GPC_LIGHTYELLOW         11
#define GPC_LIGHTBLUE           12
#define GPC_LIGHTMAGENTA        13
#define GPC_LIGHTCYAN           14
#define GPC_WHITE               15
```

- 256 colors palette file
  gpctool\sample\std.pal is the most suitable palette to express natural images in 256 colors.
  gpctool\sample\equal.pal is a palette where colors are chosen to have R(3 bits), G(3 bits), and B(2 bits) equally (including the 16 reserved colors) .

Note: For 256 colors, the rest of the index can be used freely.

Color Palette Text Format

(The values are expressed as decimal numbers delimited by a space.)

```
256          ← The first line represents the number of colors of the palette
48 48 48     ← The 1st color. The numbers starting from the second line represent data of each of R, G, and B.
48 48 112    ← The 2nd color
48 48 176    ← The 3rd color
48 48 240      .
48 112 48      .
48 112 112     .
48 112 176     .
    .          .
    .
    .
240 0 240
240 64 240   ← The 254th color
240 128 240  ← The 255th color
255 255 255  ← The 256th color
```

RGB Image Data Format

id = 0	4 bytes
size of width	4 bytes
size of height	4 bytes
dataR(1, 1)	1 byte
dataG(1, 1)	1 byte
dataB(1, 1)	1 byte
dataR(2, 1)	1 byte
dataG(2, 1)	1 byte
dataB(2, 1)	1 byte
.	.
.	.
.	.
dataR(1, 2)	
dataG(1, 2)	
dataB(1, 2)	
.	
.	
.	
dataR(Xn, Yn)	
dataG(Xn, Yn)	
dataB(Xn, Yn)	

3.1.2 Converting Image ROM Data into C Source File

Convert the completed image ROM data into a C source file for linking to your application. To perform this conversion, use "bin2s.exe" .

Example: `>bin2c.exe image.bin > image.c` (Use the Redirect function of DOS.)
`>bin2c.exe color.bin >> image.c`

In this example, convert "image.bin" into C source "image.c", and append the converted data of "color.bin" at the end of "image.c" .
The input file names "image.bin" and "color.bin" are labeled as global variable at the leading position of each converted data as the follows. (Symbol name can be altered using the "-lsymbol" option of "bin2c.exe".)

Example: Contents of "image.c"

```
const unsigned char image[] = {
0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7,
0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7,
0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7,
0xac, 0xac, 0xac, 0xac, 0xac, 0xac, 0xac, 0xac, , , ,
};
/* total 2800 bytes data */

const unsigned char color[] = {
0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f,
0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f,
0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f,
0x0f, 0x0c, 0x0c, 0x0f, 0x0f, 0x0a, 0x0a, 0x08, , , ,
};
/* total 2800 bytes data */
```

3.2 Creating an Application and Linking with the GRAPHIC33 Library

You can implement graphics function on the E0C33 chip by calls to GRAPHIC33 library functions. Note that this product contains GUI functions to provide GUI capability, and graphics functions to provide primitive drawing capabilities. This allows you to create Graphics application simply by including the functions in your program.
For more details on GUI functions and graphics functions, see Section 5, "GRAPHIC33 Library Reference" .
Refer to sample programs provided in the "gpclib\sample\" directory.

You must also include the source of the created image ROM data in your program, or link it along with the GRAPHIC33 library after C.

Information on running sample programs using the DMT33006, DMT33MON and DMT33LCD boards are provided for reference in the Appendix.

4 GRAPHIC33 Tool Reference

This section describes the functions of each GRAPHIC33 tool and how to use them.

4.1 Outline of GRAPHIC33 Tools

GRAPHIC33 tools are PC softwares used to create image ROM data, which is written on a E0C33 Family chip. All tools are 32-bit applications executable from a DOS prompt, and will run under Windows 95, Windows NT 4.0, or later Windows versions. (For more information on developing environments, refer to Section 2.1, "Developing Environment.")
All GRAPHIC33 tools and related files are located in the "gpctool" folder (directory).
The configuration of GRAPHIC33 tools and the procedure for creating image ROM data are shown in Figure 4.1.1.

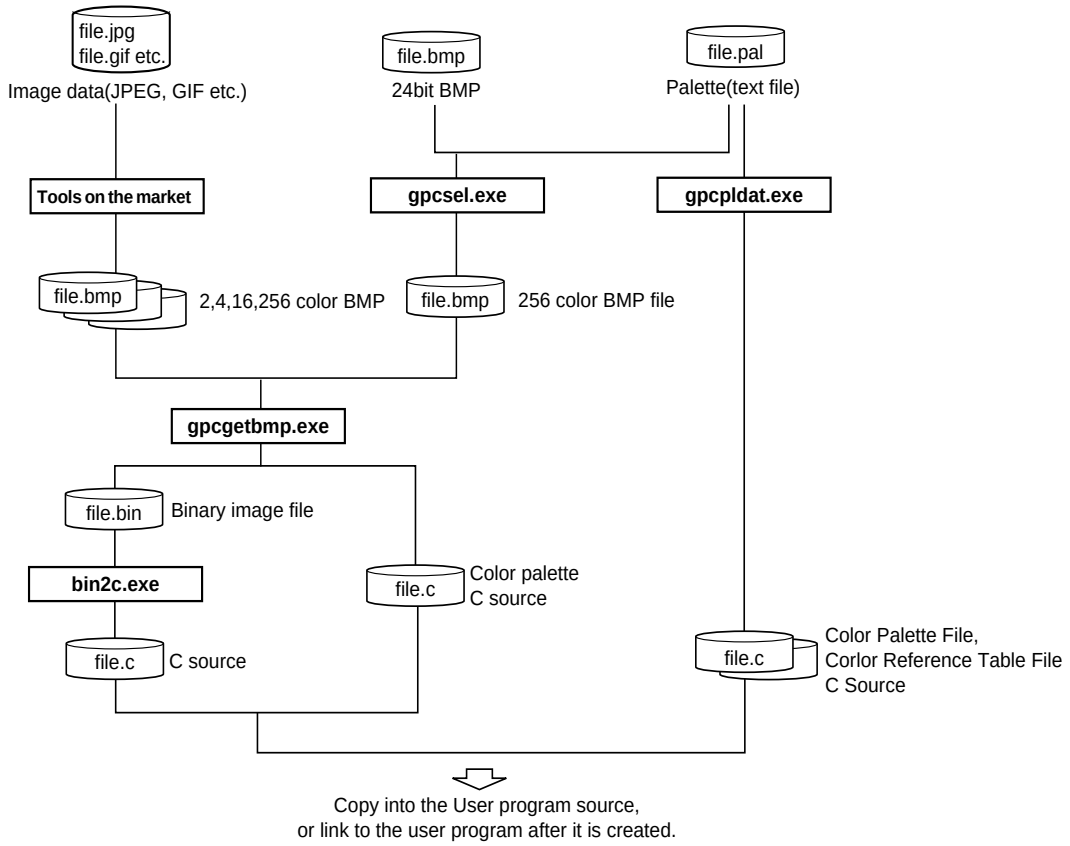


Figure 4.1.1 Flowchart for Creating Image ROM Data

Image ROM data creation tools consist of a series of programs for Conversion to Image ROM data, creating color palette. They are also used to generate C and assembly source files for the EOC33. All tools are 32-bit executable applications from a DOS prompt, and can also be used from a batch file. The tool are listed in Table 4.1.1.

Table 4.1.1 List of Image ROM Data Creation Tools

Tool	Description
gpcgetbmp.exe	Converts BMP file to BMP image data.
gpcsel.exe	Converts 24-bit bitmap file to 8-bit bitmap file (using color palette).
gpcpldat.exe	Makes C source file for color palette from color palette (text file).
bmp2rgb.exe	Converts BMP file to RGB file.
bin2c.exe	Converts Image data file to C source file.
bin2s.exe	Converts Image data fie to assembler source file.

4.2 Description of Tools

This section describes the function of each GRAPHIC33 tool and explains how each is used.
Start each tool from the DOS prompt. Usage is displayed when you start a tool without specifying parameters in the command line. In the command line explanation, [ ] denotes optional parameters (may be omitted). Parameters in italic indicate an appropriate value or file name.

Note: The file names that may be specified in each tool are limited as follows:

- File name: Up to 32 characters
- Valid characters: a to z, A to Z, 0 to 9, _, .

4.2.1 gpcgetbmp.exe

Function: Read the bitmap file in 1, 2, 4, and 8 bit, and save the 1, 2, 4, 8 bit bitmap file except the header information as bitmap image data file.

Usage:
```
DOS> gpcgetbmp.exe image256.bmp image256.bin color.c
```

Parameters:

`image256.bmp`	input	Bitmap file
`image256.bin`	output	Bitmap image data
`color.c`	output	Color palette (C source)

Example:
```
DOS> gpcgetbmp.exe image256.bmp image256.bin color.c
```
The contents of image256.bin after execution contain only image data where the header and palette are removed.
When you use the image256.bin file in programs, set it as follows.

```
DOS>bin2c.exe image256.bin > image256.c
```
Contents of image256.c after execution:
```
const unsigned char image256[] = {
0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7,
0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, ...
```

Contents of color.c after execution:
```
const unsigned char color[256*3] = {
// 0
0x00, 0x00, 0x00,  0x80, 0x00, 0x00,  0x00, 0x80, 0x00,  0x80, 0x80, 0x00,
0x00, 0x00, 0x80,  0x80, 0x00, 0x80,  0x00, 0x80, 0x80,  0xC0, 0xC0, 0xC0,
...
```

If you do not need the color table, omit it as follows before use.
```
DOS> gpcgetbmp.exe image256.bmp image256.bin
```

4.2.2 bmp2rgb.exe

Function: BMP → RGB Conversion tool
The first specified file name is a BMP form input file, and the second name is a RGB form output file.
A blank or tab character is needed between the arguments.

Usage:
```
DOS> bmp2rgb flower2.bmp flower2.rgb
```

Parameters:

`flower2.bmp`	input	BMP form file name
`flower2.rgb`	output	RGB form file name

Example:
```
DOS> bmp2rgb flower2.bmp flower2.rgb
```
Contents of flower2.rgb after execution is in the RGB format.
When you use flower2.rgbfile in programs, set it as follows.

```
DOS>bin2s -l sample flower2.rgb > bitmap.s
```
Contents of bitmap.s after execution:
```
.global   sample
.align    2
sample:
     .byte    0x00 0x00 0x00 0x00 0xa0 0x00 0x00 0x00
     .byte    0xf0 0x00 0x00 0x00 0x5d 0x4a 0x25 0x5f ...
```

Note: If the argument is not specified, a message appears to explain the usage.
If an error occurs, an error message is displayed and the output file cannot be created.

4.2.3 bin2c.exe

Function: Binary → CC33 C source conversion tool
If you specify this option, use the ' - ' and add arguments.
Use a blank or tab character to delimit the arguments.

Usage:
```
DOS> bin2c [options] infile.bin > outfile.c
```

Parameters:

`infile.bin`	input	Binary file name
`outfile.c`	output	C source file name

-1 label : Label name specification
When you want to use a symbol which different from the file name, specify the option. Available characters are a - z, A - Z, 0 - 9, and _. Specify the label name with less than 32 characters.

Example:

1) If you omit -1option, the input file name becomes a symbol name.
```
DOS>bin2c.exe image256.bin > image256.c
```
Contents of image256.c after execution:
```
const unsigned char image256[] = {
0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7,
0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7,
              .
              .
0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f,
0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f,
};
/* total 2800 bytes data */
```

2) When you want to use a symbol name which is different from the file name, specify using the -1option.
```
DOS>bin2c.exe -l sample image256.bin > image256.c
```
Contents of image256.c after execution:
```
const unsigned char sample[] = {
0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7,
0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7, 0xd7,
                .
                .
0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f,
0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f,
};
/* total 2800 bytes data */
```

Note: If the output file already exists, the file will be overwritten.
If the option specification is wrong, the usage is displayed.

4.2.4 bin2s.exe

Function: Binary → CC33 assembler source conversion tool
If you specify this option, use the ' - 'and add arguments
Use a blank or tab character to delimit the arguments.

Usage:
```
DOS> bin2s [options] infile.bin > outfile.s
```

Parameters:

`infile.bin`	Input	Binary file name
`outfile.s`	Output	Assembler source file name

-l label : Specify label name
When you want to use a symbol which different from the file name, specify the option. Available characters are a - z, A - Z, 0 - 9, and _. Specify the label name with less than 32 characters.

Example: 1) If you omit the -1 option, the input file name becomes a symbol name.

```
DOS>bin2s  flower2.rgb > bitmap.s
```

Contents of bitmap.s after execution:

```
.global    flower2
.align     2
flower2:
    .byte    0x00 0x00 0x00 0x00 0xa0 0x00 0x00 0x00
    .byte    0xf0 0x00 0x00 0x00 0x5d 0x4a 0x25 0x5f
               .
               .
    .byte    0xf8 0xff 0x3a 0x00
; total 724 bytes data
```

2) When you want to use a symbol name which is different from the file name, specify the -1 option.

```
DOS>bin2s -l sample flower2.rgb > bitmap.s
```

Contents of bitmap.s after execution:

```
.global  sample
.align   2
sample:
    .byte    0x00 0x00 0x00 0x00 0xa0 0x00 0x00 0x00
    .byte    0xf0 0x00 0x00 0x00 0x5d 0x4a 0x25 0x5f
               .
               .
    .byte    0xf8 0xff 0x3a 0x00
; total 724 bytes data
```

Note: If the output file already exists, the file will be overwritten.
If the option is wrong, the usage is displayed.

4.2.5 gpcsel.exe

Function: Read the 24-bit bitmap file, and replace the color using the specified palette.
Save it as an 8-bit bitmap file.
Regarding the color choice method, the colors are by default replaced with the most similar color in the RGB space (simple para-colors). The maximum input size is 1280 × 1024 pixels.

Usage:
```
DOS> gpcsel.exe [-d] infile.bmp outfile.bmp palle.pal
```

Parameters:

`infile.bmp`	Input	24-bit bitmap file (RGB)
`outfile.bmp`	Output	8-bit bitmap file
`pallet.pal`	Input	Color palette file

-d:Fdither option
Use the diffusion dither method to replace colors.

Example:
```
DOS> gpcsel.exe -d image.bmp image256.bmp std.pal
```

4.2.6 gpcldat.exe

Function: Read the color palette file, convert it to C source, and then output it.
Create a color reference table from the color palette, and then output it.
The table size is 12 bits (4096 bytes).

Usage:
```
DOS> gpcpldat.exe pallet.pal pallet.c search_tbl.c
```

Parameters:

`pallet pal`	Input	Color palette file (RGB)
`pallet.c`	Output	Color palette (C source)
`search_tbl.c`	Output	Color reference table (C source)

Example:
```
DOS> gpcpldat.exe std.pal pallet.c search_tbl.c
```
Contents of pallet.c after execution:
```
const int gpc_cnt_std = 0x100;
const unsigned char gpc_pal_std[] = {
0x00, 0x00, 0x00,  0x80, 0x00, 0x00,  0x00, 0x80, 0x00,  ...
;
0xff, 0xff, 0xff }
```

Contents of search_tbl.c after execution:
```
const unsigned char gpc_stbl_std[] = {
0x40, 0x40, 0x84, 0x41, 0x41, 0x41, 0x41, ...
;
0x8a, 0x0f, 0x0f }
```

Reference: The following information is stored in the color palette C source.
const unsigned char gpc_cnt_Input file name // Number of colors of color palette
const unsigned char gpc_pal_Input file name // Color palette data arrangement

The following information is stored in the color reference table.
const unsigned char gpc_stbl_Input file name // Color reference table arrangement

5 GRAPHIC33 Library Reference

This section gives a detailed explanation of GRAPHIC33 library functions, as well as precautions regarding their usage.

5.1 Outline of GRAPHIC33 Library

Functional outline

The GRAPHIC33 library is a set of graphics processing functions in srf33 library format, used after linking functions to the target application. The following functions can be executed by calling the required function from the target application:

- Primitive drawing function
  Drawing functions such as point, circle, ellipse, rectangle, arc, and fill
- Functions to display and save picture images
- Function to display text
  - Fonts for Shift JIS
    1 byte (1-byte character- numbers, alphabets, katakana)
    2 bytes (2-byte character- Chinese characters, katakana, hiragana)
- Graphical user interface function
  Window form, pop-up window, control button, image button, radio button, check box, and field function
- Key and mouse event functions

This package also contains C source files for top-level functions based on GRAPHIC33 library functions and assembly source files for initialization purposes, all or part of which may be copied for use into the target application. These function sets permit easy implementation of graphics processing functions in your system.

Program structure

Figure 5.1.1 describes the structure of the application programs.

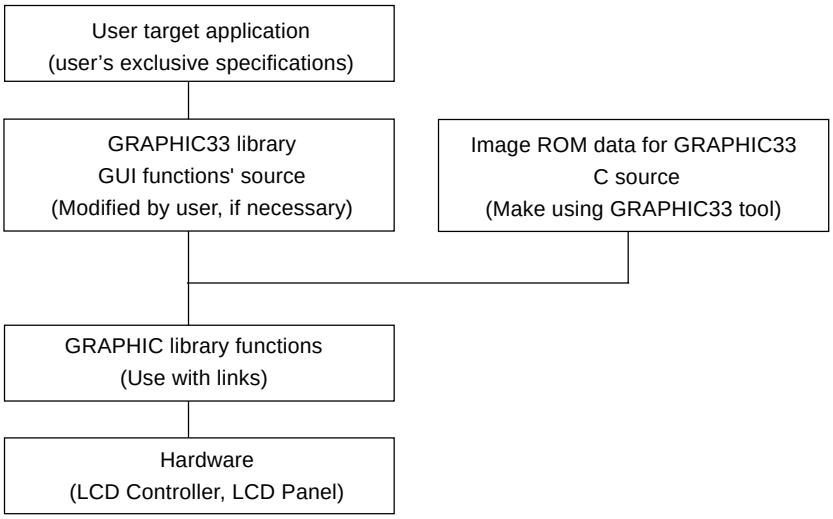


Figure 5.1.1 Program Structure

GRAPHIC33 Library Configuration

All of the GRAPHIC33 library and related files are located in the "gpclib" folder (directory). The contents of the "gpclib" folder are given below.

gpclib .....GRAPHIC33 library related items
- readmeja.txt Japanese manual
- readme.txt English manual
- **lib**.....GRAPHIC33 library directory
  - gpc33.lib GRAPHIC33 library
  - ros33.lib ROS33 library
  - gpcpal.o Palette library
- **include**.....Header file directory for GRAPHIC33 library functions
  - gpc.h Header file of graphics library
  - gpcdrv.h Header file of display driver
  - gpcgui.h Header file for GUI functions
  - c33208.h Header file for C33208 symbol definition
  - itron.h Header file of ROS33
  - ros33.h Header file of ROS33
  - gpcpal.h Header file for palette functions
- **src**.....GUI source directory
  - gpcgui.c GUI functions
  - gpcevent.c Event functions
  - guiimage.c GUI image data (cursor, mouse, UP button, DOWN button)
- **libsrc**.....Library source directory
  - gpcdrv.c Display driver
  - gpcfont1.c Graphics font (1 byte)
  - gpcfont2.c Graphics font (2 bytes)
  - gpccolor.c Color table
- **sample**.....Sample directory

  (For details on the sample programs ’structure and their use, refer to the "readme.txt" or "readmeja.txt" file in "gpclib".)
  - **gpc** .....Graphics function sample directory
  - **gpc_k** .....GUI key sample directory
  - **gpc_m** .....GUI mouse sample directory
  - **gpcpal** .....Palette sample directory
  - **gpcpal2** .....Palette sample directory (No.2)

∗ The configuration for the GUI function and Graphic function is described later.

5.2 Hardware Resources

Hardware resource GUI use

When you use GUI, use the hardware resource listed below.

- K50 to K54 port, and all their related control registers
- 16-bit programmable timer (timer3), and all their related control registers

Operating clock

- This library inputs 20MHz oscillation to high-speed (OSC3) clock frequency.
  20MHz is doubled to 40MHz using PPL.
  Also, the bus clock runs at 20 MHz for 40MHz × 1/2.
- Clock for the built- in LCD controller (CLKI), OSC3 (20MHz) is set to be used in 1/3 cycle.

Amount of memory and stack

The GRAPHIC33 library uses the following memory sizes:

Table 5.2.1 Amount of memory of the gpc33.lib library

*1 If pop-up window is not used, BSS(RAM) of the GUI system takes about 3 KBytes.

Object	Code(ROM)	BSS(RAM)	Stack
GPC related function (gpc33.lib)	15KByte	500Byte	170Byte
GUI related function (gpcgui.o, gpcevent.o)	4.2KByte	23KByte *1	250Byte
1 byte font (gpcfont1.o in gpc33.lib)	3KByte	—	—
2 byte font (gpcfont2.o in gpc33.lib)	164KByte	—	—
Display driver (gpcdrv.o in gpc33.lib)	1KByte	16Byte	—

5.3 GUI Function

GUI functions in the "gpcgui.c", "gpcevent.c" are C source files used to help implement each function, and are made using GRAPHIC33 library functions. Table 5.3.1 lists the GUI functions.

Table 5.3.1 List of GUI Function

src\gpcgui.c

Function name	Description
guiDrawButton( )	Draws button
guiDrawCheckBox( )	Draws check box
guiGroupRadioButton( )	Selects one from group
guiDrawRadioButton( )	Draws radio button
guiEraseRadioButton( )	Erases radio button
guiDrawForm( )	Draws form window
guiLoadForm( )	Loads and draw form window
guiPopupWindow( )	Draws pop-up window
guiClosePopupWindow( )	Erases pop-up window
guiMouseButtonDown( )	Presses down button using the mouse
guiMouseButtonUp( )	Raises up button using the mouse
guiButtonDown( )	Reverses button
guiButtonUp( )	Releases pressed button
guiCheckPos( )	Checks mouse position
guiCheckScrollButton( )	Checks scroll button
guiCheckControlExE( )	Checks control button for execution
guiDispCursor( )	Displays cursor
guiEraseCursor( )	Erases cursor
guiDispMouse( )	Displays mouse
guiEraseMouse( )	Erases mouse
guiSetCursorImage( )	Sets cursor image
guiSetMouseImage( )	Sets mouse image
guiSetMemoryBuf( )	Sets memory buffer of pop-up window
guiSetCursorBuf( )	Sets buffer of mouse and cursor
guiDrawField( )	Draws text field
guiScrollField( )	Scrolls text field
guiSetFieldText( )	Sets initialization of text field

Table 5.3.2 List of Event Function

src\gpcevent.c

Function name	Description
guiGetCursor( )	Gets key code
guiInitEvent( )	Initializes event
guiApplication( )	Checks event form execution judgment
guiGetEvent( )	Gets event message
guiTM16CH3_GetKey( )	Sends event message
guiTM16CH3_INT( )	Event generation interrupt routine
guiSetKeyTimer( )	Initializes and runs event interrupt
guiKillKeyTimer( )	Stops event interrupt

5.3.1 Graphical User Interface Structure

Header file for the graphical user interface is "gpcgui.h" in the "\include" directory.

Declaration of GUI define

```
#define GUI_TAB_SIZE        4
```

∗ Tab size is 4 by default.
You can change the GUI_TAB_SIZE as follow:
Example:

```
#define GUI_TAB_SIZE        8

#define GUI_SPACE           " "  // SPACE Code
#define GUI_ENTER           '\n' // CR Code
#define GUI_TAB             0x9  // TAB Code

#define GUI_TRUE            1    // TRUE
#define GUI_FALSE           0    // FALSE

#define GUI_UP_SCROLL       (-1) // SCROLL UP
#define GUI_DOWN_SCROLL     1    // SCROLL DOWN
#define GUI_MAX_SCROLL      10   // MAX SCROLL
```

Declaration of GUI structure

Declaration of form structure

```
typedef struct _guiForm {
    short x;              // Position x of Form
    short y;              // Position y of Form
    short width;          // Width of Form
    short height;         // Height of Form
    short id;             // Form ID
    short control_cnt;    // Number of Control in the Form
    char  *text;          // Pointer to Tittle of Form
    guiControl *control;  // Pointer to Control in the Form
} guiForm;
```

The x, y, width, and height represent position, width, and height of the form respectively.
The id is the ID of form. Be careful not to have duplicate IDs.
The control_cnt is the number of controls in the form.
The "text" is the title of the form. It appears on the title bar of the form.
The control is a pointer to control in the form.
The guiControl is a control structure.
For details on control structure, refer to "Declaration of Control Structure" below.

Declaration of Control Structure

Declare the control used in the window form.

```
typedef struct _guiControl {
    short x;                    // Position x of Control in the Form
    short y;                    // Position y of Control in the Form
    short width;                // Width of Control
    short height;               // Height of Control
    short id;                   // Control ID in the Form
    char  style;                // Control Style
    char  *text;                // Pointer to Control Text
    guiControlAttr *controlAttr; // Pointer to Control Attribute
    guiControlGroup *group;     // Pointer to Control Group
    char  *Image;               // Pointer to Image Button
} guiControl;
```

The x, y, width, and height are the position, width and height of the form respectively.
The id is the ID of Form. Be careful not to have duplicate IDs.
The "style" is the type of the control. The types are GPC_BUTTON, AGPC_SCROLL_BUTTON, AGPC_CHECK_BOX, AGPC_RADIO_BUTTON, and AGPC_FIELD.
The text is the title of the control. It is displayed on the control.
The controlAttr is the control attribute's structure.
The group is the control group structure. Refer to "Declaration of Attribute structure" below.
The Image is the image point of the button. bit map image is displayed on the button. Set the image size to 24×14.

Declaration of Group Structure

Declare the control group.
∗ When you want to use more than two groups in one screen, declare different group structures.
Group declaration applies only to radio button and check box.
Group structure declaration displays the group frame and group title.
Radio button and check box can be used without group.
If group is not used, set as group member of control to "0".

```
typedef struct _guiControlGroup {
    short x;         // Group Position of X in the Form
    short y;         // Group Position of Y in the Form
    short width;     // Group Width in the Form
    short height;    // Group Height in the Form
    short groupID;   // Control Group ID in the Form
    char  *title;    // Point to Control Group Title in the Form
} guiControlGroup;
```

The x, y, width, and height represent the position, width, and height of the form.
The groupID is ID of the group. Be careful not to have duplicate IDs.
The "title" is the title of the control group. It is displayed on the group.

Declaration of Attribute Structure

Declare the attribute of radio button and check box.
Attribute value: (1:ON, 0:OFF)

```
typedef struct _guiControlAttr {
    char  attr;      // Control's Attribute in the Control
} guiControlAttr;

guiControlAttr Ctrl2_Attr[3]; // Changable Control Declare
```

∗ Be sure to initialize before using the control. Set only one radio button to ON.

```
    Ctrl2_Attr[0] = 0;  // Radio1
    Ctrl2_Attr[1] = 1;  // Radio2
    Ctrl2_Attr[2] = 0;  // Radio3
```

Declaration of enumeration model of control

```
typedef enum _guiControlStyle {
    GPC_BUTTON,
    GPC_SCROLL_BUTTON,
    GPC_CHECK_BOX,
    GPC_RADIO_BUTTON,
    GPC_FIELD
} guiControlStyle;
```

GPC_BUTTON is a button, GPC_SCROLL_BUTTON is a scroll button, GPC_CHECK_BOX is a check box, GPC_RADIO_BUTTON is a radio button, and GPC_FIELD is a text field.

5.3.2 Event Structure

To process the mouse and keyboard, create an event and send the event message. Then, get and process the message from the event processing routine.

Event header file is "gpcgui.h" in the "\include" directory.

GUI Define Declaration

Declaration of the Event Key

This is a key code of SW7 of DMT33006.

∗ If you have modified the key code on DMT33006, rewrite according to the key code before use.

```
//Event Key Code for DMT33006
  #define GUI_KEY_UP    1  //  0x01
  #define GUI_KEY_LEFT  2  //  0x02
  #define GUI_KEY_DOWN  4  //  0x04
  #define GUI_KEY_RIGHT 8  //  0x08
  #define GUI_KEY_UL    3  //  0x03  Right at the middle between UP and LEFT
  #define GUI_KEY_DL    6  //  0x06  Right at the middle between DOWN and LEFT
  #define GUI_KEY_DR    12 //  0x0c  Right at the middle between DOWN and RIGHT
  #define GUI_KEY_UR    9  //  0x09  Right at the middle between UP and RIGHT
  #define GUI_KEY_ON    16 //  0x10
```

Declaration of the event message define

∗ Declare the value of message avoiding duplication.

```
//Events Code for KEY
  #define GUI_MSG_KEYON  16   // 0x10
  #define GUI_MSG_KEYOFF 32   // 0x20

//Events Code for KEY & MOUSE
  #define GUI_MSG_MOVE   110
```

GUI Event Declaration

Declaration of Event Structure

This is a structure for the use of event processing.

```
typedef struct _guiEvent {
    short     Events;       // Types of Events
    short     KeyCode;      // Key Code
    short     CtrlID;       // Current Ctrl ID
    short     CtrlNo;       // Current Ctrl No
    short     FormModifiers; // Form Execute Status (GUI_TRUE:1, GUI_FALSE:0)

    int       screenX;      // Current Mouse Position X in the Form
    int       screenY;      // Current Mouse Position Y in the Form

} guiEvent;
```

Use the event structure for the event processing at the application.
The "Events" are event messages. Event messages are GUI_MSG_KEYON, GUI_MSG_KEYOFF, and GUI_MSG_MOVE.
The "KeyCode" is key code. Key codes are GUI_KEY_UP, GUI_KEY_LEFT, GUI_KEY_DOWN, GUI_KEY_RIGHT, GUI_KEY_UL, GUI_KEY_DL, GUI_KEY_DR, GUI_KEY_UR, and GUI_KEY_ON.
The "CtrlID" is the current ID of control. Declare and use control and make sure there are no duplicate controls in the ID form of the control.
The "CtrlNo" is a number for the current control. The numbers of controls are set in a sequential order.
The "FormModifiers " is a form execution flag to determine whether to execute other forms. Form execution flags are GUI_TRUE (execute) and GUI_FALSE (do not execute).
The "screenX" is the X-coordinate of the current mouse position.
The "screenY" is the Y-coordinate of the current mouse position.

Declaration of position structure

This structure is used for checking the mouse position.

```
typedef struct _gpcPoint {
    int  x;     // Current Position X
    int  y;     // Current Position Y
}gpcPoint;
```

The x is the X-coordinate of the current mouse position.
The y is the Y-coordinate of the current mouse position.

5.3.3 GUI Function

guiDrawButton( )

Function: Draws button

Format: `int guiDrawButton(guiControl *control);`

Parameters: `guiControl *control` Structure pointer to the current control

Return Value: `GPC_ERR_OK` Terminated normaly

Description: Draws a button for the current control.

guiDrawCheckBox( )

Function: Draws checkbox

Format: `int guiDrawCheckBox(guiControl *control);`

Parameters: `guiControl *control` Structure point of the current control

Return value: `GPC_ERR_OK` Terminated normaly

Description: Draws a check box for the current control.

guiGroupRadioButton( )

Function: Selects only one radio button from a group

Format: `int guiGroupRadioButton(guiControl *control, int pos);`

Parameters: `guiControl *control` Structure pointer of the current control
`int pos` Position of the current control

Return value: `GPC_ERR_OK` Terminated normaly

Description: Selects only one control you pressed from a group.

guiDrawRadioButton( )

Function: Draws radio button

Format: `int guiDrawRadioButton(guiControl *control);`

Parameters: `guiControl *control` Structure pointer of the current control

Return value: `GPC_ERR_OK` Terminated normaly

Description: Draws radio button for the current control

guiEraseRadioButton( )

Function: Erases radio button

Format: `int guiEraseRadioButton(guiControl *control);`

Parameters: `guiControl *control` Structure pointer of the current control

Return value: `GPC_ERR_OK` Terminated normaly

Description: Erases the radio button except the currently pressed control.

guiDrawForm( )

Function: Draws Form

Format: `int guiDrawForm(guiForm *form);`

Parameters: `guiForm *form` Structure pointer to the current form

Return value: `GPC_ERR_OK` Terminated normaly

Description: Draws the specified current form.
Controls are all included in the form.
∗ The form frame is drawn with black color.
Black is used for the background color of the title, and white is used for text.

guiLoadForm( )

Function: Draws form and set event

Format: `int guiLoadForm(guiForm *form, void (*event_handler)(int event));`

Parameters:

`guiForm *form`	Structure pointer to the current form
`void (*event_handler)(int event)`	Function pointer of the event to process

Return value: `GPC_ERR_OK` Terminated normaly

Description: Initializes the memory and draw the specified form, then set the event.
Call guiDrawForm( )in this function.

guiPopupWindow( )

Function: Draws Pop-up window

Format: `int guiPopupWindow(guiForm *form);`

Parameters: `guiForm *form` Structure pointer to the current form

Return value: `GPC_ERR_OK` Terminated normaly

Description: Draws the pop-up window for the rectangle area.
Sets the buffer using guiSetMemoryBuf( ). Stores the background of pop-up window in this buffer.
To close the pop-up window, use guiClosePopupWindow( ).
Be sure to set only one general button for the control of the pop-up window.
∗ Frame color of the pop-up window is drawn with black color.
Black is used for the background color of the title, and white is used for text.

guiClosePopupWindow( )

Function: Erases pop-up window

Format: `int guiClosePopupWindow(guiForm *form);`

Parameters: `guiForm *form` Structure pointer to the current form

Return value: `GPC_ERR_OK` Terminated normaly

Description: Closes the pop-up window of the specified rectangular area.
Note that this closes the last drawn pop-up window using guiPopupWindow( ). Be careful before using it.

guiMouseButtonDown( )

Function: Down the button using the mouse

Format: `int guiMouseButtonDown(guiPoint *Point);`

Parameters: `guiPoint *Point` Current position of the mouse

Return value: `GPC_ERR_OK` Terminated normaly

Description: Erases the mouse on the current button, reverses the button pushed by the current form, and then shows the mouse.

guiMouseButtonUp( )

Function: Up the button using the mouse

Format: `int guiMouseButtonUp(guiPoint *Point);`

Parameters: `guiPoint *Point` Current position of the mouse

Return value: `GPC_ERR_OK` Terminated normaly

Description: Erases the mouse on the current button, reverses the button pushed by the current form, and then shows the mouse.
Takes off the pushed button.

guiButtonDown( )

Function: Down button

Format: `int guiButtonDown(guiForm *form, int pos);`

Parameters: `guiForm *form` Structure pointer to the current form
`int pos` Position of the current form

Return value: `GPC_ERR_OK` Terminated normaly

Description: Inverts the button pushed by the current Form.

guiButtonUp( )

Function: Up button

Format: `int guiButtonUp(guiForm *form, int pos);`

Parameters: `guiForm *form` Structure pointer to the current form
`int pos` Position of the current form

Return value: `GPC_ERR_OK` Terminated normaly

Description: Inverts the button pushed by the current Form.

guiCheckPos( )

Function: Checks the mouse position

Format: `int guiCheckPos(guiForm *form, gpcPoint point);`

Parameters:
`guiForm *form`	Structure pointer to the current form
`gpcPoint point`	Position of the current mouse

Return value:
`iRetVal`	Number of the position of the mouse

Description: Checks the button position of the control of the current form.
Point is the position obtained from the mouse event.
Return value is the number of the button of the mouse position.

guiCheckScrollButton( )

Function: Checks the scroll button

Format: `int guiCheckScrollButton(guiForm *form, int pos);`

Parameters:
`guiForm *form`	Structure pointer to the current form
`int pos`	Position of the current control

Return value:
`iRetVal`	Scroll button	`GUI_TRUE`
	Not scroll button	`GUI_FALSE`

Description: Checks whether the control number of the current form is a scroll button or not.
Point is the position obtained from the mouse event.
Return values are the number of the current mouse position.

guiCheckControlExE( )

Function: Checks the control button

Format: `int guiCheckControlExE(guiForm *form, int controlID);`

Parameters:
`guiForm *form`	Structure pointer to the current form
`int controlID`	Current control ID

Return value:
`iRetVal`	Scroll button	`GUI_TRUE`
	Not scroll button	`GUI_FALSE`

Description: Checks whether the current control ID of the current form is a button or not.
Return value:
	Button	GUI_TRUE
	Not Button	GUI_FALSE

guiDispCursor( )

Function: Displays cursor

Format: `int guiDispCursor(guiForm *form, int pos);`

Parameters:
`guiForm *form`	Structure pointer to the current form
`int pos`	Position of the current control

Return value:
`GPC_ERR_OK`	Terminated normaly

Description: Displays the cursor at the position of the current control in the current form.

guiEraseCursor( )

Function: Erases Cursor

Format: `int guiEraseCursor(guiForm *form, int pos);`

Parameters:
`guiForm *form`	Structure pointer to the current form
`int pos`	Position of the former control

Return value: `GPC_ERR_OK` Terminated normaly

Description: Erases the cursor at the position of the former control in the current form.

guiDispMouse( )

Function: Displays mouse

Format: `int guiDispMouse(gpcPoint *Point);`

Parameters:
`gpcPoint *Point`	Current mouse position

Return value: `GPC_ERR_OK` Terminated normaly

Description: Displays the mouse cursor at the position of the mouse in the current form.

guiEraseMouse( )

Function: Erases Mouse

Format: `int guiEraseMouse(gpcPoint *Point);`

Parameters:
`gpcPoint *Point`	Former mouse Position

Return value: `GPC_ERR_OK` Terminated normaly

Description: Erases the cursor of the former mouse in the current form.

guiSetCursorImage( )

Function: Sets cursor image

Format: `int guiSetCursorImage(unsigned char *pCursor);`

Parameters:
`unsigned char *pCursor`	Cursor image point

Return value: `GPC_ERR_OK` Terminated normaly

Description: Sets the specified cursor image to the cursor global variable.
Set the cursor image size to 12×18.

guiSetMouseImage( )

Function: Sets mouse image

Format: `int guiSetMouseImage(unsigned char *pMouse, unsigned char *pMouseMask);`

Parameters:
`unsigned char *pMouse`	Mouse image point
`unsigned char *pMouseMask`	Mouse mask image point

Return value: `GPC_ERR_OK` Terminated normaly

Description: Sets the specified mouse image and mouse mask to the mouse global variables.
Set the mouse image size to 12×18.

guiSetMemoryBuf( )

Function: Sets buffer of pop-up window

Format: `int guiSetMemoryBuf(unsigned char *pMemoryBuf);`

Parameters: `unsigned char *pMemoryBuf` Memory buffer point

Return value: `GPC_ERR_OK` Terminated normaly

Description: Sets the memory buffer which restore the pop-up window to the global variable.
In the case of 8BPP, the size means width × height .
Example: If the pop-up window size is 130 × 150 dots as 8BPP, then the pMemoryBuf size is 130 × 150=19500 bytes.

guiSetFieldText( )

Function: Initializes text field

Format: `int guiSetFieldText(guiControl *field, unsigned char *text, unsigned char *FieldBuf);`

Parameters:

`guiControl *field`	Current text field pointer
`unsigned char *text`	Data pointer displayed on text field
`unsigned char *FieldBuf`	Buffer pointer which text field scroll copies

Return value: `GPC_ERR_OK` Terminated normaly

Description: Sets the data of text field specified using guiControl structure to match the text field.
Sets the text data pointer.
FieldBuf size is (font height + 4) × text field width (pixel).

guiDrawField( )

Function: Draws text field

Format: `int guiDrawField(guiControl *field);`

Parameters: `guiControl *field` Current text field pointer

Return value: `GPC_ERR_OK` Terminated normaly

Description: Draws the specified text field using the guiControl structure.

guiScrollField( )

Function: Scrolls text field

Format:
```
int guiScrollField(guiControl *field, int scroll, int
ScrollLines);
```

Parameters:

`guiControl *field`	Current text field pointer
`int scroll`	Scroll value to set
`int ScrollLines`	Number of lines to scroll

Return value: `GPC_ERR_OK` Terminated normaly

Description: Scrolls the specified text field using the guiControl structure.
If scroll is UP_SCROLL, then scroll up the number of lines of text field.
If scroll is DOWN_SCROLL, then scroll down the number of lines of text field.
ScrollLines is the number of lines to scroll. The maximum number of lines to scroll is 10.

5.3.4 GUI Event Function

guiGetCursor( )

Function: Gets key code

Format: `unsigned char guiGetCursor( );`

Parameters: None

Return value: K50 to K54 key codes of DMT33006

Description: Gets the K50 to K54 key codes from DMT33006.
Return values are the pressed key codes.

guiInitEvent ( )

Function: Initializes event

Format: `void guiInitEvent (void);`

Parameters: None

Return value: None

Description: Initializes event.

guiApplication( )

Function: Determines whether to perform event form

Format: `int guiApplication(void);`

Parameters: None

Return value: `GUI_TRUE` Perform
`GUI_FALSE` Do not perform

Description: Gets the determine flag of event structure.
Determines from guiCurEvent.FormModifiers.
(Perform:GUI_TRUE, Do not Perform:GUI_FALSE)

guiGetEvent( )

Function: Gets event

Format: `void guiGetEvent(guiEvent* Event);`

Parameters: `guiEvent* Event` Event structure pointer

Return value: None

Description: Gets the message from the mailbox, and sets the mouse event message to the event structure.

guiTM16CH3_GetKey( )

Function: Gets key code and sends event message

Format: `void guiTM16CH3_GetKey( );`

Parameters: None

Return value: None

Description: Gets the key codes and sends the event message to the mailbox.

guiTM16CH3_INT( )

Function: Interrupt routine

Format: `void guiTM16CH3_INT( );`

Parameters: None

Return value: None

Description: Interrupt routine for the key and mouse event.

guiSetKeyTimer( )

Function: Initializes and starts interrupt

Format: `void guiSetKeyTimer(unsigned short freq);`

Parameters: `freq` Interval time on interrupt

Return value: None

Description: Initializes and starts interrupt of the key and mouse event.
CPU Clock 40MHz.
Default standard time of freq ≒ 1 millisecond.

guiKillKeyTimer( )

Function: Stops interrupt

Format: `void guiKillKeyTimer( );`

Parameters: None

Return value: None

Description: Stops interrupt of the key and mouse event.

5.4 GRAPHIC33 Library Functions

GRAPHIC33 library "gpc.lib" includes functions to process primitive drawing, image display, and text display. You can use drawing, image display, and text display functions by linking "gpc.lib" to your application.
" Figure 5.4.1" shows a list of GRAPHIC33 library.

Table 5.4.1 GRAPHIC33 library function list

gpc33.lib

Function name	Description
gpcInitGc( )	Sets Graphic initialization
gpcSetColor( )	Sets Drawing Color
gpcGetColor( )	Gets Drawing Color
gpcSetBkColor( )	Sets Background Color
gpcGetBkColor( )	Gets Background Color
gpcSetLineWidth( )	Sets Line Width
gpcGetLineWidth( )	Gets Line Width
gpcSetLineStyle( )	Sets Line Style
gpcGetLineStyle( )	Gets Line Style
gpcSetFont( )	Sets Font
gpcGetFont( )	Gets Font
gpcDrawPoint( )	Draws Point
gpcDrawLine( )	Draws Line
gpcDrawRect( )	Draws Rectangle
gpcFillRect( )	Fills Rectangle
gpcInvertRect( )	Inverts Rectangle
gpcDrawEllipse( )	Draws Ellipse
gpcFillEllipse( )	Fills Ellipse
gpcDrawCircle( )	Draws Circle
gpcFillCircle( )	Fills Circle
gpcDrawArc( )	Draws Arch
gpcDrawText( )	Draws Text
gpcPutImage( )	Displays Image
gpcGetImage( )	Saves Image
gpcGetVersion( )	Gets Library Version

gpc33.lib, libsrc\gpcdrv.c

Function name	Description
seInit( )	Initializes Display Driver
seRegisterDevice( )	Sets SED1375 Register
seSetBitsPerPixel( )	Sets Bit/Pixel
seGetBitsPerPixel( )	Gets Bit/Pixel
seGetBytesPerScanline( )	Gets Byte Number of Line
seGetScreenSize( )	Gets Screen Size
seGetDisplayMem( )	Gets Display Memory Address
seGetDisplayMemSize( )	Gets Display Memory Size
seSetLut( )	Sets Look-up Table(LUT)
seGetLut( )	Gets Look-up Table(LUT)
seSetLutEntry( )	Sets Value on Specified Register of Look-Up Table(LUT)
seGetLutEntry( )	Gets Value on Specified Register of Look-Up Table(LUT)
seSetReg( )	Sets Value on Specified Register
seGetReg( )	Gets Value of Specified Register

lib\gpcpal.o

Function name	Description
gpcMakeSearchTable( )	Makes Search Table
gpcInitClosestColor( )	Initial Setting for Color Replacing
gpcGetClosestColor( )	Selects Most Similar Color in RGB
gpcDitherInit( )	Initialization Setting of Diffusion Dither
gpcGetDitherColor( )	Selects Color Using Diffusion Dither

The specification of each function is explained below. For the actual example, refer to the source of GUI Function and Graphic Function.

5.4.1 Graphic Structure

About Graphic Structure

The gpcGCValues is a graphic control structure to control memory, color, line, and font.
The header file of graphics is "gpc.h" in the "\include"directory.

Graphic Structure

```
typedef struct _gpcGCValues {
    unsigned char *GCDispMem;    // display memory start address
    int BytesPerScanline;        // byte per scanline
    int GCColor;                 // current color
    int GCBkColor;               // Background Color for gpcDrawText
    char GCLineWidth;            // current line width
    char GCLineStyle;            // current line style
    unsigned char *GCFont1;      // point of font 1Byte
    char GCFont1Width;           // current 1byte width
    char GCFont1Height;          // current 1byte height
    unsigned char *GCFont2;      // point of font 2Byte
    char GCFont2Width;           // current 2byte width
    char GCFont2Height;          // current 2byte height
    unsigned char *GCTableEntry; // 2Byte Font Table Entry.
    char GCTableSize;            // 2Byte Font Table Entry Size.
} gpcGCValues;
```

GCColor Color

`1bit/pixel` Select from look-up table number 0, 1
`2bit/pixel` Select from look-up table number 0, 1, 2, 3
`4pit/pixel` Select from look-up table number 0, 1...14, 15
`8pit/pixel` Select from look-up table number0, 1, 2...254, 255

GCBkColor background Color of Text

`1bit/pixel` Select from look-up table number 0, 1
`2bit/pixel` Select from look-up table number 0, 1, 2, 3
`4pit/pixel` Select from look-up table number 0, 1, 2...14, 15
`8pit/pixel` Select from look-up table number 0, 1, 2...254, 255

GCLineStyle Line Style

`#define GPC_SOLID_LINE`	0	Solid Line
`#define GPC_DASH_LINE`	1	Dashed Line
`#define GPC_DOT_LINE`	2	Dotted Line
`#define GPC_DASHDOT_LINE`	3	Repeat of Dashed and Dotted Line

Example)

Set to the graphic structure directly

```
extern gpcGCValues gpcGC;
gpcGC.GCLineStyle = GPC_DOT_LINE;
```

Use Graphic Function

```
gpcSetLineStyle(GPC_DOT_LINE);
```

∗ After using the GPC_DOT_LINE setting in your appplication, you have to set the GPC_SOLID_LINE because the graphic function draws with the current GPC_DOT_LINE setting.

GCLineWidth Width of line

```
#define GPC_NORM_WIDTH      1       1 pixel width
#define GPC_THICK_WIDTH     3       3 pixel width
```

Example)

Set to the graphic structure directly

```
extern gpcGCValues gpcGC;
gpcGC.GCLineWidth = GPC_THICK_WIDTH;
```

Use Graphic Function

```
gpcSetLineWidth(GPC_THICK_WIDTH);
```

∗ After using the GPC_THICK_WIDTH setting in your application, you have to set the GPC_NORM_WIDTH, because the graphic function draws with the current GPC_THIC_WIDTH setting.

GCFont Font

Use 1-byte and 2-byte bitmap font.
Character code of 1- byte font uses ASCII, and character code of 2-byte font uses Shift JIS. The standard font size is 8 × 11dots(1 byte) and 16 × 11 dots(2 bytes).
Set the width of font(XSize) in multiples of 8.
Font structure is FONTX format. For more details, refer to "5.4.2 Structure of Font File".

∗ When you use fonts, be sure to set the new FONTX font, by calling gpcSetFont (1ByteFont, 2ByteFont) as follows.

Example)

```
// GPCFONT11A is 1-byte font
const unsigned char GPCFONT11A[] = {
  0x46, 0x4f, 0x4e, 0x54, 0x58, 0x32, 0x20, 0x20,
  0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x08, 0x0b, ... }
// GPCFONT11K is 2-byte font
const unsigned char GPCFONT11K[] = {
  0x46, 0x4f, 0x4e, 0x54, 0x58, 0x32, 0x20, 0x20,
  0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x10, 0x0b, ... }

gpcSetFont(GPCFONT11A, GPCFONT11K);
```

5.4.2 Structure of Font File

Font file of FONTX enables you to define the character size or character code freely. Also, because the character size and the character code must be used simultaneously, it is composed of a header part containing character size and character code, and a pattern part containing bit patterns. The font file structures are explained below.
∗ In this library, character code of 2-byte font is fixed as Shift JIS.

ASCII Font

The following table is the header of ASCII font.

Font header

Index	Size	Description	
0	6	Identifier	: The default is "FONTX2 " is put here
6	8	FontName	: Font name (8 bytes)
14	1	XSize	: Width of font in pixel
15	1	YSize	: Height of font in pixel
16	1	CodeType	: 0

Here, Index represents the offset from the head in the font file, and Size represents the size in bytes of the item.

Font Pattern Layout

```
(X*0+0)'th byte (X*0+1)'th byte          ......... (X*0+X-1)'th byte
(X*1+0)'th byte (X*1+1)'th byte          ......... (X*1+X-1)'th byte
                       .
                       .
                       .
(X*(Y-1)+1)'th byte  (X*(Y-1)+2)'th byte ......... (X*(Y-1)+X)'th byte
```

This pattern shows the order of a font data of XSize*YSize lines.
The upper left part represents the upper left part of the font pattern, and the lower right part represents the lower right part of the font pattern.
X refers to the byte number of 1 scanline of the font pattern [X=int((XSize-1)/8)+1], and Y is the number of scanline of the font pattern [Y=YSize] .
If the XSize of the font cannot be divided by 8, put 0 in the extra bit on the right.
For ASCII font, the font patterns are in sequential order from ASCII CODE 0x00 to 0xFF.

Character code table (2 byte font)

In the case of 2-byte font, it must match the structure of Shift JIS. Therefore, the position font exists is controlled by the table entry of the Block structure.
Tnum of Index 17 is the entry block number of 2-byte font.
And the entry character code follows in the Block structure after it.
The Block structure includes the character code (2 bytes) which the first block starts, and the character code (2 bytes) which the first block ends.
It continues to the last block in the pattern.
The Block structure is listed below.

Index	Size	Description	
17	1	Tnum	: Number of table entries
18	2	BlockStart1	: Character code with which the first area starts
20	2	BlockEnd1	: Character code with which the first are ends
		.	
		.	
		.	
N*2+16	2	BlockStartN	: Character code with which the Nth area starts
N*2+18	2	BlockEndN	: Character code with which the Nth area ends
		.	
		.	
		.	
Tnum*2+16	2	BlockStartTnum	: Character code with which the last area starts
Tnum*2+18	2	BlockEndTnum	: Character code with which the last area ends

Font patterns continues after this table.
Format of the font pattern is the same as the ASCII font.
The fonts are arranged in the order written in the area table.

5.4.3 Display Driver Structure

About the Display Driver of SED1375

SED1375 includes the start address of the display driver, size of the display driver, and Structure(HAL_STRUCT) which register access.
Make SED1375 data using the structure of the SED1375 display driver, and use it.

Structure of Display Driver

The header file of the display driver is "gpcdrv.h" in the "\include" directory.

```
typedef struct tagHalStruct
{
    unsigned long    dwDispMem;         // Start Address of VRAM
    unsigned long    dwDispMemSize;     // Size of VRAM
    unsigned long    dwControlMem;      // Start Address of control
                                           register
    unsigned char    Reg[MAX_REG + 1];  // 1375 Registers
} HAL_STRUCT;

typedef HAL_STRUCT * LPHAL_STRUCT;      // Point of 1375 display driver structure
```

Set LCD Controller Register

This section explains about the register setting of the SED1375LCD controller built into E0C332L01. It covers only the setting values related to the LCD panel. For more details, refer to the "E0C332L01 Technical Manual".

For DMT33LCD26
160 × 240dots screen, colors, D-TFT panel
12bits data interface
8bits/pixels, 256 colors

Table 5.4.3.1 Setting values related to LCD panel (for DMT33LCD26)

Register	Value	Description
0x01	0xa3	Selects D-TFT, Color Panel, Data Size 12bits
0x02	0xc0	Selects 256 colors
0x03	0x3	Selects Normal Power Running
0x04	0x13	Horizontal Display Size(Reg[0x4] +1)*8=160pixels
0x05	0xef	Vertical Display Size Reg[0x5] [0x6] +1=240lines
0x06	0x0	
0x07	0x3	FPLINE Start Position
0x08	0x1a	Horizontal non-Displaying Cycle
0x09	0x4	FPFRAME Start Position
0x0a	0xa	Vertical non-Displaying Cycle

ForDMT33LCD37
320 × 240dots screen, colors, D-TFT Panel
12bits data interface
4bits/pixels, 16 colors

Table 5.4.3.2 Setting values related to LCD panel (For DMT33LCD37)

Register	Value	Description
0x01	0xa3	Selects D-TFT, Color Panel, Data Size 12bits
0x02	0x80	Selects 16 colors
0x03	0x3	Selects Normal Power Running
0x04	0x27	Horizontal Display Size(Reg[0x4] +1)*8=320pixels
0x05	0xef	Vertical Display Size Reg[0x5] [0x6] +1=240lines
0x06	0x0	
0x07	0x3	FPLINE Start Position
0x08	0x6	Horizontal non-Displaying Cycle
0x09	0x4	FPFRAME Start Position
0x0a	0xa	Vertical non-Displaying Cycle

Sample for LF26 Panel

Create the structure of the SED1375 display driver as follows:
∗ This structure's data is defined in "main.h".
Example) 160 × 240dots screen, 8bits/pixels, 256 colors

```
const HAL_STRUCT LF26reginfo =
{
    0x380000,     /* VRAM Address */
    0x9fff,       /* VRAM Size */
    0x39ffe0,     /* Control Register */

    0x00,  0xA3,  0xC0,  0x03,  0x13,  0xEF,  0x00,  0x03,
    0x1A,  0x04,  0x0A,  0x00,  0x00,  0x00,  0x00,  0x00,
    0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,
    0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00
};
```

∗ The current start address of VRAM is 0x380000, VRAM size is 0x9fff and the start address of VRAM control is 0x39ffe0.
∗ For more details on the SED1375 settings, refer to the SED1375 manual.

5.4.4 Graphic Processing Functions

gpcInitGc( )

Function: Initializes Graphic Library

Format: `int gpcInitGc( );`

Parameters: None

Return value: `GPC_ERR_OK` Terminated normaly

Description: Initializes Memory, Color, Width, and Style used for the Graphics Library.

Default color	: GPC_BLACK(Black)
Default background color	: GPC_WHITE(White)
default Width	: GPC_NORM_WIDTH(Normal line width)
Default Style	: GPC_SOLID_LINE(Solid line)

gpcSetColor( )

Function: Sets Color

Format: `int gpcSetColor(int color);`

Parameters: `int color` Color to set the current drawing color

Return value: `GPC_ERR_OK` Terminated normaly

Description: Sets the current drawing color.

gpcGetColor( )

Function: Gets color

Format: `int gpcGetColor(int *color);`

Parameters: `int *color` Color to set the current drawing color

Return value: `GPC_ERR_OK` Terminated normaly

Description: Gets the current drawing color.

gpcSetBkColor( )

Function: Sets the background color

Format: `int gpcSetBkColor(int color);`

Parameters: `int color` Color to set the current background color

Return value: `GPC_ERR_OK` Terminated normaly

Description: Sets the color of the current background. This function is used when drawing text.

gpcGetBkColor( )

Function: Gets background color

Format: `int gpcGetBkColor(int *color);`

Parameters: `int *color` Pointer to get the current background color

Return value: `GPC_ERR_OK` Terminated normaly

Description: Gets the current background color.

gpcSetLineWidth( )

Function: Sets line width

Format: `int gpcSetLineWidth(char width);`

Parameters: `char width` Value to set the current line width

Return value: `GPC_ERR_OK` Terminated normaly

Description: Sets the current line width.

gpcGetLineWidth( )

Function: Gets background color

Format: `int gpcGetLineWidth(char *width);`

Parameters: `char *width` Pointer to get the current line width

Return value: `GPC_ERR_OK` Terminated normaly

Description: Gets the current line width.

gpcSetLineStyle( )

Function: Sets line style

Format: `int gpcSetLineStyle(char style);`

Parameters: `char style` Value to set the current line style

Return value: `GPC_ERR_OK` Terminated normaly

Description: Sets the current line style.

gpcGetLineStyle( )

Function: Gets line style

Format: `int gpcGetLineStyle(char *style);`

Parameters: `char *style` Point to get the current line style

Return value: `GPC_ERR_OK` Terminated normaly

Description: Gets the current line style.

gpcSetFont( )

Function: Sets font

Format: `int gpcSetFont(unsigned char *font1, unsigned char *font2);`

Parameters:
`unsigned char *font1` Pointer to the current 1-byte font
`unsigned char *font2` Pointer to the current 2-byte font

Return value:
`GPC_ERR_OK` Terminated normaly
`GPC_ERR_FONT` `No FONTX2 format`

Description: Sets the current 1-byte and 2-byte fonts.
Checks the font to set using Identifier of the "FONTX2" form font.

gpcGetFont( )

Function: Gets font

Format: `int gpcGetFont(unsigned char *font1, unsigned char *font2);`

Parameters:
`unsigned char *font1` Pointer of the current 1-byte font
`unsigned char *font2` Pointer of the current 2-byte font

Return value: `GPC_ERR_OK` Terminated normaly

Description: Gets the current 1-byte and 2-byte fonts.

gpcDrawPoint( )

Function: Draws point

Format: `int gpcDrawPoint(int x, int y);`

Parameters:
`int x` Position x of the current point
`int y` Position y of the current point

Return value: `GPC_ERR_OK` Terminated normaly

Description: Draws the point at the position of (x, y) using the current drawing color.
If x, y are not the values in the screen, the function does not draw.
∗ If the coordinates are out of the horizontal or vertical size set by the display driver, the function does not draw.

gpcDrawLine( )

Function: Draws line

Format: `int gpcDrawLine(int x1, int y1, int x2, int y2);`

Parameters:
`int x1` Position x of the current line start
`int y1` Start position y of the current line
`int x2` End position x of the current line
`int y2` End position y of the current line

Return value: `GPC_ERR_OK` Terminated normaly

Description: Draws a solid line between the specified two points, (x1, y1) and (x2, y2), using the current drawing color, line style, and line width.
∗ If the coordinates are out of the horizontal or vertical size set by the display driver, the function does not draw.

gpcDrawRect( )

Function: Draws rectangle

Format:
```
int gpcDrawRect(int left, int top, int right, int bottom);
```

Parameters:

`int left`	Left corner position of the current rectangle
`int top`	Upper corner position of the current rectangle
`int right`	Right corner position of the current rectangle
`int bottom`	Bottom corner position of the current rectangle

Return value: `GPC_ERR_OK` Terminated normaly

Description: Draws a rectangle using the current drawing color, line style, and line width. The (left, top) specifies the upper left corner, and the (right, bottom) specifies the right bottom corner of the rectangle.
∗ If the coordinates are out of the horizontal or vertical size set by the display driver, the function does not draw.

gpcFillRect( )

Function: Fills rectangle

Format:
```
int gpcFillRect(int left, int top, int right, int bottom);
```

Parameters:

`int left`	Left corner position of the current rectangle
`int top`	Upper corner position of the current rectangle
`int right`	Right corner position of the current rectangle
`int bottom`	Bottom corner position of the current rectangle

Return value: `GPC_ERR_OK` Terminated normaly

Description: Fills the rectangle using the current drawing color.
The (left, top) specifies the upper left corner, and the (right, bottom) specifies the right bottom corner of the rectangle.
∗ If the coordinates are out of the horizontal or vertical size set by the display driver, the function does not draw.

gpcInvertRect( )

Function: Inverts rectangle

Format:
```
int gpcInvertRect(int left, int top, int right, int bottom);
```

Parameters:

`int left`	Left corner position of the current reversed area
`int top`	Upper corner position of the current reversed area
`int right`	Right corner position of the current reversed area
`int bottom`	Bottom corner position of the current reversed area

Return value: `GPC_ERR_OK` Terminated normaly

Description: Reverses and displays the bitmap of the specified rectangle area.
The (left, top) specifies the upper left corner, and the (right, bottom) specifies the right bottom corner of the rectangle.
∗ If the coordinates are out of the horizontal or vertical size set by the display driver, the function does not draw.

gpcDrawEllipse( )

Function: Draws Ellipse

Format: `int gpcDrawEllipse(int left, int top, int right, int bottom);`

Parameters:

`int left`	Left corner position of the ellipse
`int top`	Upper corner position of the ellipse
`int right`	Right corner position of the ellipse
`int bottom`	Bottom corner position of the ellipse

Return value: `GPC_ERR_OK` Terminated normaly

Description: Draws an ellipse using the current drawing color.
The (left, top) specifies the upper left corner, and the (right, bottom) specifies the right bottom corner of the rectangle. This function draws an ellipse which go into this rectangle using the current drawing color.
GCLineStyle, GCLineWidth parameter do not affect the arc, circle, and ellipse.
∗ If the coordinates are out of the horizontal or vertical size set by the display driver, the function does not draw.

gpcFillEllipse( )

Function: Fills ellipse

Format: `int gpcFillEllipse(int left, int top, int right, int bottom);`

Parameters:

`int left`	Left corner position of the current ellipse
`int right`	Upper corner position of the current ellipse
`int right`	Right corner position of the current ellipse
`int bottom`	Buttom corner position of the current ellipse

Return value: `GPC_ERR_OK` Terminated normaly

Description: Fills an ellipse using the current drawing color.
The (left, top) specifies the upper left corner, and the (right, bottom) specifies the right bottom corner of the rectangle.
This function fills the ellipse which go into this rectangle using the current drawing color.
∗ If the coordinates are out of the horizontal or vertical size set by the display driver, the functions do not draw.

gpcDrawCircle( )

Function: Draws circle

Format: `int gpcDrawCircle(int x, int y, int radius);`

Parameters:

`int x`	Position x of the current circle
`int y`	Position y of the current circle
`int radius`	Current radius

Return value: `GPC_ERR_OK` Terminated normaly

Description: Draws a circle with (x, y) as the center with the value of "radius" as the radius, using the current drawing color.
∗ If the coordinates are out of the horizontal or vertical size set by the display driver, the functiondoes not draw.
Drawing does not occur if r(radius) <=0.

gpcFillCircle( )

Function: Fills circle

Format: `int gpcFillCircle(int x, int y, int radius);`

Parameters:

`int x`	x position of the current circle
`int y`	x position of the current circle
`int radius`	current radius

Return value: `GPC_ERR_OK` Terminated normaly

Description: Fills a circle with (x, y) as the center with the value of "radius", using the current drawing color.
∗ If the coordinates are out of the horizontal or vertical size set by the display driver the function does not draw.
Drawing does not occur if r(radius)<=0.

gpcDrawArc( )

Function: Draws arc

Format: `int gpcDrawArc(int x, int y, int angle1, int angle2);`

Parameters:

`int x`	x position of the current arc
`int y`	y position of the current arc
`int angle1`	Current start angle
`int angle2`	Current end angle

Return value: `GPC_ERR_OK` Terminated normaly

Description: Draws an arc with (x, y) as the center, with the value of "radius" as the radius, using the current drawing color.
Arc is from angle1 to angle2. If angle1 is 0 and angle2 is 360, the function draws a perfect circle. The angle is counted counterclockwise with 3 o'clock position as 0 degree, and 12 o'clock position as 90 degrees. angle1 must be larger than angle2.
GCLineStyle, GCLineWidth parameters do not affect arc, circle, and ellipse.
∗ If the coordinates are out of the horizontal or vertical size set by the display driver, the function does not draw.
Drawing does not occur if r(radius)<=0.

gpcDrawText( )

Function: Displays text

Format: `int gpcDrawText(int x, int y, char *text, int length);`

Parameters:

`int x`	Position x of the current text
`int y`	Position y of the current text
`char *text`	Current text pointer
`int length`	Current text size

Return value: `GPC_ERR_OK` Terminated normaly

Description: This function outputs text with the coordinates (x, y) on the screen as the left upper corner of the first text line.
text is the text line to be output, and length is the number of characters to be output.
This function enables change of the background color of the text.
Set the color to use for the line using gpcSetBkColor( ) function as described below, before using gpcDrawText( ).

Example)
```
gpcGetBkColor(&BkColor);
gpcSetBkColor(GPC_LIGHTBLUE);
gpcDrawText(40, 30, "Graphic Text!", 14);
gpcSetBkColor(BkColor);
```

∗ Be sure to return the background color to the default setting as the background color set by gpcSetBkColor (GPC_LIGHTBLUE) affects the next text.
∗ Extra text is not displayed.
If the text is displayed on the border between inside and outside of the screen, then the y coordinate position is adjusted so that the text does not go beyond the screen.
Adjustment, if it occurs, is as follows:

If -11 < y < 0, y = 0
If y < -11, do not display.
If Height of screen< y < Height of screen + Height of font, then y = y - Height of font.
Text located on the x coordinate borders is not displayed.

gpcPutImage( )

Function: Displays image

Format:
```
int gpcPutImage(int x, int y, int width, int height, unsigned char *buf,
int op);
```

Parameters:

`int x`	Position x of the current image
`int y`	Position y of the current image
`int width`	Width of the current image
`int height`	Height of the current image
`unsigned char *buf`	Current image pointer
`int op`	Current image option

Return value: `GPC_ERR_OK` Terminated normaly

Description: This function outputs a bit image, with the coordinates (x, y) on the screen as the left upper corner of the bit image. The image indicates an area in the memory where the bit image is stored. The op argument of putimage indicates a combination operation based on the pixels already existing on the screen and their matching source pixels in the memory.
GPC_COPY_PUT 1 : Copy
GPC_XOR_PUT 2 : Exclusive-or
GPC_OR_PUT 3 : Logical sum
GPC_AND_PUT 4 : Logical product
GPC_NOT_PUT 5 : Reversing copy of source
The gpcPutImage does not display the image if it is bigger than the one allocated.
∗ Buf, width, and height parameters of gpcPutImage are restricted as follows:
If the values of Buf, width, and height written from gpcGetImage differ from Buf, width, and height of gpcPutImage, then window display cannot be guaranteed.

gpcGetImage( )

Function: Stores image

Format:
```
int gpcGetImage(int x, int y, int width, int height, unsigned char
*buf);
```

Parameters:

`int x`	Position x of the current image
`int y`	Position y of the current image
`int width`	Width of the current image
`int height`	Height of the current image
`unsigned char *buf`	Current image pointer

Return value: `GPC_ERR_OK` Terminated normaly

Description: Copys image from the display memory to the memory.
The x , y and width, height define the rectangle on the screen to be copied.
The buffer indicates an area in the memory where a bit image is stored.
The gpcGetImage does not store the image if it is bigger than the one allocated.

gpcGetVersion( )

Function: Gets version

Format: `int gpcGetVersion(void);`

Parameters: None

Return value: Version of the library

Description: Gets the version of the library.
For example, if the version is 1.0, the function returns 0x10.

5.4.5 Display Driver Function

seInit( )

Function: Initializes LCD controller block

Format: `int seInit(void);`

Parameters: None

Return value: `GPC_ERR_OK` Terminated normaly

Description: Initializes the SED1375 LCD controller block
Register of BCU are set as follows:
1)Area 6 - 4 configuration register (0x4812A)
- Area 6 output disable delay time = 0.5 cycle
- Area 6 wait number = 2 wait

2)Bus controller register (0x4812E)
- Select external interface method = A0 method
- #WAIT Enable

3)Access control register (0x48132)
Area 6 internal access
4)BCLK selection register (0x4813A)
Select BCU_CLK

seRegisterDevice( )

Function: Sets register of SED1375s

Format: `int seRegisterDevice(const LP_HAL_STRUCT lpHalInfo);`

Parameters: `const LP_HAL_STRUCT lpHalInfo` Structure pointer of display driver

Return value: `GPC_ERR_OK` Terminated normaly

Description: Sets the SED1375 register value of the LP_HALSTRUCT model to the register of SED1375.
Be sure to read it at the beginning of the program.

seGetBitsPerPixel( )

Function: Gets BPP

Format: `int seGetBitsPerPixel(int *pBitsPerPixel);`

Parameters: `int *pBitsPerPixel` Point of BPP in the current color
Parameter return value: BPP(1, 2, 4, 8)

Return value: `GPC_ERR_OK` Terminated normaly

Description: Gets the BPP of the current color according to the register of SED1375.

seSetBitsPerPixel( )

Function: Sets BPP

Format: `int seSetBitsPerPixel(int BitPerPixel);`

Parameters: `int BitsPerPixel` Color to set BPP
Parameter value: BPP(1, 2, 4, 8)

Return value: `GPC_ERR_OK` Terminated normaly

Description: Sets the BPP of the current color to SED1375.

seGetBytesPerScanline( )

Function: Gets the bytes of line

Format: `seGetBytesPerScanline(int *pBytePerScanline);`

Parameters: `int *pBytePerScanline` Pointer to the byte number of line
Parameter return value: Number of byte

Return value: `GPC_ERR_OK` Terminated normaly

Description: Gets the bytes of the line in the current mode.

seGetScreenSize( )

Function: Gets the width and height size of the screen

Format: `int seGetScreenSize(int *width, int *height);`

Parameters: `int *width` Pointer to the width size
`int *height` Pointer to the height size
Parameter return value: width size, height size

Return value: `GPC_ERR_OK` Terminated normaly

Description: Gets the width pixel size and height pixel size of the current screen.

seGetDisplayMem( )

Function: Gets display memory address

Format: `unsigned char *seGetDisplayMem( );`

Parameters: None

Return value: start address pointer of the memory

Description: Gets the display memory address
Start Address is defined directly in the header file of the display driver (main.h).

```
const HAL_STRUCT LF26reginfo =
{
0x380000,     /* VRAM Address */
;
```

seGetDisplayMemSize( )

Function: Gets display memory size

Format: `int seGetDisplayMemSize( );`

Parameters: None

Return value: Memory size

Description: Gets the display memory size.

seSetLut( )

Function: Sets Look-Up Table(LUT)

Format: `int seSetLut(unsigned char *pLUT, int nCount);`

Parameters:

`unsigned char *pLUT`	Pointer to a color table to set the Look-Up Table (LUT)
`int nCount`	Number of colors to set the Look-Up Table (LUT)

Return value: `GPC_ERR_OK` Terminated normaly

Description: Sets a color table to the Look-Up Table (LUT)

seGetLut( )

Function: Gets Look-Up Table(LUT)

Format: `int seGetLut(unsigned char *pLUT, int nCount);`

Parameters:

`unsigned char *pLUT`	Pointer to a color table to get the Look-Up Table (LUT)
`int nCount`	Number of colors to get from the Look-Up Table (LUT)

Return value: `GPC_ERR_OK` Terminated normaly

Description: Gets a color table from the Look-Up Table (LUT)

seSetLutEntry( )

Function: Sets the index of Look-Up Table(LUT)

Format: `int seSetLutEntry(int index, unsigned char *pLUT);`

Parameters:

`unsigned char *pLUT`	Pointer to a color table to set in the index position of the Look-Up Table (LUT)
`int index`	index position to set on the Look-Up Table (LUT)

Return value: `GPC_ERR_OK` Terminated normaly

Description: Sets a color table at the index position of the Look-Up Table (LUT)

seGetLutEntry( )

Function: Sets the index of Look-Up Table(LUT)

Format: `int seGetLutEntry(int index, unsigned char *pLUT);`

Parameters:

`unsigned char *pLUT`	Pointer to a color table to get from the index position of the Look-Up Table (LUT)
`int index`	index position of the Look-Up Table (LUT)

Return value: `GPC_ERR_OK` Terminated normaly

Description: Gets the color table at the index position of the look-up table(LUT).

seSetReg( )

Function: Sets the value to the specified register

Format: `int seSetReg(int index, unsigned char value);`

Parameters:

`int index`	index position of the register
`unsigned char value`	value to set at the index position of the register

Return value: `GPC_ERR_OK` Terminated normaly

Description: Sets a value at the index position of the register.

seGetReg( )

Function: Gets the value of the specified register

Format: `unsigned char seGetReg(int index);`

Parameters: `int index` index position of the register

Return value: `unsigned char value` Value at the index position of the register

Description: Gets the value at the index position of the register.

5.4.6 Palette Functions

gpcMakeSearchTable( )

Function: Makes a search table.

Format:
```
int gpcMakeSearchTable(unsigned char *pallet, int size, unsigned char
*stable);
```

Parameters:

`unsigned char *pallet`	Color palette (RGB color)
`int size`	Palette colors (maximum 256 colors)
`unsigned char *stable`	Buffer for storing color search table

Return value: Size of color search table (4096 bytes)

Description: This function makes a color search table for the specified color palette.
Size of the table is 12 bits (4096 bytes).

gpcInitClosestColor( )

Function: Sets up initialization for color replacement.

Format:
```
void gpcInitClosestColor(unsigned char *pallet, int size,
unsigned char *stable);
```

Parameters:

`unsigned char *pallet`	Color palette (RGB color)
`int size`	Palette colors (maximum 256 colors)
`unsigned char *stable`	Color search table

Return value: None

Description: Sets up initialization for color replacement.
gpcGetClosestColor( ) and gpcGetDitherColor( ) select color from the color palette specified here.
The color search table uses the gpcMakeSearchTable( ) function or the 12-bit table created with gpcpldat.exe.

gpcGetClosestColor( )

Function: Selects the closest color in the RGB space.

Format:
```
int gpcGetClosestColor(unsigned char *src, int size, unsigned char
*dst);
```

Parameters:

`unsigned char *sr`	Input data (RGB color)
`int size`	Number of input pixels
`unsigned char *dst`	Output data buffer (index color)

Return value: `GPC_ERR_OK` Terminated normaly

Description: This function loads the RGB data specified by src, selects the closest color in the RGB space, and writes an index (size 1 byte) for the color palette to the output data buffer. For the input data, each pixel has 3 bytes of data (1 byte each of R, G, and B in that order).

gpcDitherInit( )

Function: Sets up initialization for using the diffusion dithering method.

Format: `void gpcDitherInit(short *pBuf, int size);`

Parameters:

`short *pBu`	Pointer to work buffer
`int size`	Work buffer size

Return value: None

Description: This function sets up the initialization for color replacement using the diffusion dithering method. gpcInitClosestColor( ) is also required.
Also, if the diffusion dithering method is used in color selection, then a work buffer must be set up.
The size of the buffer can be calculated using the following formula:
(Maximum width of image + 2) × 6

gpcGetDitherColor( )

Function: Selects color using the diffusion dithering method.

Format: `int gpcGetDitherColor(unsigned char *src, int width, unsigned char *dst);`

Parameters:

`unsigned char *src`	Input data (24-bit RGB)
`int width`	Width of image (number of pixels per line)
`unsigned char *dst`	Pointer to output data buffer

Return value:

`GPC_ERR_OK`	Terminated normaly
Error(0x0)	

Description: This function loads a line of RGB data specified by src, selects a color using the diffusion dithering method, and writes an index (size 1 byte) for the color palette to the output data buffer. For the input data, each pixel has 3 bytes of data (1 byte each of R, G, and B in that order).

5.5 Program Examples

This section describes some cautions to take notice of when creating Graphic33 application programs using the sample programs in the "sample\" directory.
Also, it includes cautions on creating emulation programs on window.
For information on handling software development tools and on C source creation methods, refer to the "E0C33 Family C Compiler Package Manual."

5.5.1 Graphic Function Samples

boot.s file

Set up the boot.s file as shown in the following code.

```
#define SP_INI 0x2000 ; sp is in end of 8KB internal RAM
#define GP_INI 0x0000 ; global pointer %r8 is 0x0

BOOT:
    xld.w   %r8,SP_INI
    ld.w    %sp,%r8    ; set SP
    ld.w    %r8,GP_INI; set global pointer
    xcall   InitBusCtrl     ; Initialize BCU
    xcall   main       ; go to main
END:
    nop
    jp END
```

The main function in "main.c" located in the "sample\gpc\src" directory is listed here.

Rules for Main function

Example)

```
#include "gpc.h"        // Graphic33 library header
#include "gpcdrv.h"     // Display Driver header
#include "main.h"       // main sample header
```

- Sample program's data declaration is defined in "main.h".

```
#define CPU_IO_TOP 0x40000   // For C33208 I/O top
```

- Main function

```
int main (void)
{
  int   iRet;
  extern unsigned char GPCFONT11A[];  // 1Byte Fontx in the gpcfont1.c
  extern unsigned char GPCFONT11K[];  // 2Byte Fontx in the gpcfont2.c

  // initialize control block of SED1375
  seSetInit(CPU_IO_TOP);

  // Initialize 1375 register for LF26
  seRegisterDevice((HAL_STRUCT *)&LF26reginfo);
  // Set Look-up Table 8bpp default color
  seSetLut(gpcCOLOR8, 256);

  // Initializes parameters and resources used by Graphic33
  iRet = gpcInitGc( );
  if(iRet != GPC_ERR_OK) return iRet;

  // Initialize Font
  gpcSetFont(GPCFONT11A, GPCFONT11K);

  ------------------------------------------------------------
  Create user program here
  ------------------------------------------------------------

} // End Of Main( )
```

- In the main function, be sure to always call seSetInit(CPU_IO_TOP) at first, then followed by seRegisterDevice( ), seSetLut( ) and gpcInitGc( ) in that order.
- seSetInit( ) initializes the control block of SED1375.
- seRegisterDevice( ) sets up SED1375Register.
- seSetLut( ) sets up the look-up table.
- gpcInitGc( ) is a function which initializes parameters and resources used by gpc33.
- If font is used, set gpcSetFont(GPCFONT11A and GPCFONT11K).

GPCFONT11A is the variable for 1-byte font, and GPCFONT11K is the variable for 2-byte font.
Example: If an application do not use 2-byte font, set to 0 (zero) as follows :

```
#define NO_FONT2   0
gpcSetFont(GPCFONT11A, NO_FONT2);
```

- Also, include the "gpc.h", "gpcdrv.h", and "main.h" files before using it.

5.5.2 GUI Key and Mouse Event Samples

This section describes how to handle events.
Key and the mouse cannot be used at the same time.

- When using a key event, refer to the samples in the key sample directory "sample\gui_k".
- When using a mouse event, refer to the samples in mouse sample directory "sample\gui_m".
- Key and mouse use SW7's K50(↑), K51(←), K52(↓), K53(→), and K54(Enter).

vector.s file

Set up the vector.s file as shown in the following code.

```
.word  Boot  ;  0 Reset
;
.word  int_dispatch  ;  12 Software Exception 0
```

Interrupt routine for when ROS33 is used

```
;
.word  guiTM16CH3_INT  ;  42 16-bit Timer #3 compare B
```

Interrupt routine for when key and mouse events are used.

```
;
```

boot.s file

Set up the boot.s file as shown in the following code.

```
#define SP_INI   0x00002000 ; sp is in end of 8KB internal RAM
#define PSR_INIT 0x00000110 ; InitIntr. Level 1, Intr. enable

.global Boot
Boot:
    xld.w   %r4,  SP_INI
    ld.w    %sp,  %r4
    xld.w   %r4,  PSR_INIT
    ld.w    %psr, %r4  ; set PSR
    xcall   InitBusCtrl     ; Initialize BCU
    xcall   main ; goto main
```

The main function in "main.c" located in the "sample\gui_m\lf26" directory is listed here.

Rules for Main function

Example)

```
  #include "gpc.h"    // Graphic33 library header
  #include "gpcgui.h" // GUI header
  #include "gpcdrv.h" // Display Driver header
  #include "ros33.h"  // ROS header
  #include "main.h"   // Main sample header
```

- Sample program's data declaration is defined in "main.h".

```
#define CPU_IO_TOP 0x40000    // For C33208 I/O top
```

Main function

```
int main (void)
  {
     int  iRet;
     extern unsigned char GPCFONT11A[];  // 1Byte Fontx in the gpcfont1.c
     extern unsigned char GPCFONT11K[];  // 2Byte Fontx in the gpcfont2.c
     ------------------------------------------------------------
     1. Graphic33's basic initialization
     ------------------------------------------------------------
     seSetInit(CPU_IO_TOP);
     seRegisterDevice((HAL_STRUCT *)&LF26reginfo);
     seSetLut(gpcCOLOR8, 256);
     iRet = gpcInitGc( );
     if(iRet != GPC_ERR_OK) return iRet;
     gpcSetFont(GPCFONT11A, GPCFONT11K);
     ------------------------------------------------------------
     2. Initialize mouse event and create form
     ------------------------------------------------------------
     guiInitEvent( );
     iRet = initForm( );
     if(iRet != GPC_ERR_OK) return iRet;
     guiInitMouse( );
     ------------------------------------------------------------
     3. ROS33 settings
     ------------------------------------------------------------
     // initialize system & key interrupt
     sys_ini( );  // in ros33 library
     guiSetKeyTimer(6);  // 6ms

     // create task guiEventLoop & idle_task in ros33 library
     vcre_tsk(1, guiEventTask,   1, (UW)&(stack1[STACK_SIZE]));
     vcre_tsk(2, guiAnimateTask, 2, (UW)&(stack2[STACK_SIZE]));
     vcre_tsk(3, guiIdle_task,   3, (UW)&(idle_stack[STACK_SIZE]));

     // Start task guiEventLoop & idle_task in ros33 library
     sta_tsk(1, 0);
     sta_tsk(3, 0);

     // Start System interrupt in ros33 library
     sys_sta( );

  }  // End Of Main( )
```

- Within the main function, be sure to call seSetInit(CPU_IO_TOP) at first, followed by seRegisterDevice( ), seSetLut( ), and gpcInitGc( ) in that order.
- When using ros33, call sys_ini( ) first and sys_sta( ) last in setting ros33.
- Also, include the following files: "gpc.h", "gpcdrv.h", "gpcgui.h", "ros33.h", and "main.h".
- guiInitEvent( ) sets up event initialization.
- initForm( ) creates form and then initializes form.
- guiInitMouse( ) sets up mouse initialization.
- guiSetKeyTimer(6) sets the mouse timer to 6 ms.

Tasks

The library ros33.lib used in Graphic33 limits use of up to 3 tasks. Therefore, the task IDs is from 1 to 3.
For the executing task, use vcre_tsk( ) within the main function to define it.
Operation of a system call using an ID not defined here is not guaranteed.
In the above main( ) example, it initially defines guiEventTask.
Example)`vcre_tsk(1, guiEventTask,1, (UW)&(stack1[STACK_SIZE]));`
It defines the task ID (1st argument), task function guiEventTask(2nd argument), priority1(3rd argument), and start address stack1 (4th argument).
When defined, the task is in an idle state. To start it, use sta_tsk( ).

Example)`sta_tsk(1, 0);`
Because the task ID (1st argument) and the task start code are not used in ROS33, set them to 0.

∗ sta_tsk(2, 0) is defined when animation is started.
With sys_sta( ), the system starts the multitasking environment.

∗ For details on ROS33, refer to the ROS33 manual.

Idle task

To handle the case where a task in executable state does not exist, it is necessary to prepare an idle task within the user program.
For the interrupt, set acceptability and priority to the lowest level.
Be sure to start it in main( ) in advance.
Define guiIdle_task in "main.c".
Also, do not return from the idle task.

```
void guiIdle_task( )
{
    while(1);
}
```

* For details on ROS33, refer to the ROS33 manual.

Event handling sequence

1. guiTM16CH3_INT( ) Event interrupt occurs
2. guiTM16CH3_GetKey( ) Event interrupt routine
   - guiGetCursor( ) Get key code of SW7
   - snd_msg(1, msg) Send event message
3. Event tasks
   Each individual task is created as a function in the following way. However, the tasks do not handle returning value.
   Example) guiEventTask( ) defines an event task at "main.c".

```
void guiEventTask( void )
  {
  guiEvent Event;
  while(1) {
       guiGetEvent(&Event);
       guiEventProcess(&Event);
       if(guiApplication( )){
            guiFormHandle( );
            }
       }
  }
```

   - `guiGetEvent(&Event)`
     Gets event message and sets it up at event structure.
   - `guiEventProcess(&Event)`
     Handles each event. This is the user program section.
   - `guiApplication( )`
     Gets run flag of event form.
     This is decided at guiCurEvent.FormModifiers. (GUI_TRUE: execute; GUI_FALSE: do not execute)
   - `guiFormHandle( )`
     This is the event handler function pointer. It handles the current form's event function.
     If it is set in the following way, then guiFormHandle handles the Form1HandleEvent( ) function.
     Example) `guiFormHandle = Form1HandleEvent;`

Key event handling

This section describes the "keyevent.c" file located in the key sample directory "sample\gui_k\src".
For all of the source, refer to the "keyevent.c" file.

```
guiEventProcess(&Event)
{
    short    iEvent,  iCode;

    // Follow event structure to handle each event
    iEvent = Event->Events;
    iCode  = Event->KeyCode;

    switch (iEvent){
        case GUI_MSG_KEYON:
            // Handle key ON event
            // Reversal handling if current button is pressed
            break;
        case GUI_MSG_KEYOFF:
            // Handle key OFF event
            // Reversal handling if current button is released
            break;
        case GUI_MSG_MOVE:
            // Handle key MOVE event
            // Handle key cursor display according to key's code if a key is moved
            break;
     }
}
```

Mouse event handling

This section describes the "mouse.c" file located in the mouse sample directory "sample\gui_m\lf26".
For all of the source, refer to the "mouse.c" file.

```
guiEventProcess(&Event)
{
    short    iEvent,  iCode;

    // Follow event structure to handle each event
    iEvent = Event->Events;
    iCode  = Event->KeyCode;

    switch (iEvent){
        case GUI_MSG_KEYON:
            // Handle mouse ON event
            // Reversal handling if current button is pressed
            break;
        case GUI_MSG_KEYOFF:
            // Handle mouse OFF event
            // Reversal handling if current button is released
            break;
        case GUI_MSG_MOVE:
            // Handle mouse MOVE event
            // Handle mouse display according to mouse's code if mouse is moved
            break;
     }
}
```

Key event handler

This section describes the "main.c" file located in the key sample directory "sample\gui_k\src".
For all of the source, refer to the "main.c" file.

```
Form1HandleEvent( )
{
    switch(guiCurEvent->CtrlID) {
        case ID_POPUP:
            // Displays user pop-up window
            guiPopupWindow((guiForm *)&form_popup);
            // Sets pointer of current pop-up windows to the global variable
            guiCurForm = (guiForm *)&form_popup;
            // Sets pointer of current event handler to the global variable
            guiFormHandle = Form3HandleEvent;
            // Sets controller's position to display key cursor for current form
            guiCurEvent->CtrlNo = 0;
            // Displays current key cursor
            // guiCurEvent->CtrlNo is controller's number for displaying key cursor
            guiDispCursor(guiCurForm, guiCurEvent->CtrlNo);
            break;
        case ID_OPTION:
        ;
    }
}
```

Mouse event handler

This section describes the "main.c" file located in the mouse sample directory "sample\gui_m\lf26".
For all of the source, refer to the "main.c" file.

```
Form1HandleEvent( )
{
    guiPoint iPoint;
    // Get current mouse co-ordinates to display mouse
    iPoint.x = guiCurEvent->screenX;
    iPoint.y = guiCurEvent->screenY;

    switch(guiCurEvent->CtrlID) {
        case ID_POPUP:
            // Displays user pop-up window
            guiPopupWindow((guiForm *)&form_popup);
            // Sets pointer of current window to the global variable
            guiCurForm = (guiForm *)&form_popup;
            // Sets pointer of current event handler to the global variable
            guiFormHandle = Form3HandleEvent;
            // Displays current mouse
            guiDispMouse((guiPoint *)&iPoint);
            break;
        case ID_OPTION:
        ;
    }
}
```

About the text field settings

The scroll function is included to display text string and view multiple sets of image data.
Scroll can be UP scroll and DOWN scroll.
Each scroll can be up to 10 lines.
2-byte font cannot be used in the text field.
If text field is used, set it in the way as shown in the following code.
Set guiSetFieldText at "main.c".
Example)

```
guiSetFieldText((guiControl *)&control5, (unsigned char *)Text,
(unsigned char *)&FieldBuf);
```

- Text field form data for display in text field

```
const guiControl control5[] = {
10,  24, 140, 182, ID_FIELD,   GPC_FIELD,(char *)0,
(guiControlAttr *)0,(guiControlGroup *)0, (unsigned char *)0,,,}
```

- Data for display in text field

```
const unsigned char Text[] = "    The E0C332L01 is a CMOS 32-bit
microcomputer composed of a CMOS 32-bit RISC core,"
"ROM, RAM, DMA, timers, SIO, PLL, LCDC and other circuits.¥n"
"The E0C332L01 can be operated with high speed and spend little
current.¥n"
"With the ADC, PWM and the MAC function, the E0C332L01 is suitable
for voice applications and PDAs.";
```

- Text field form data for display in text field

```
static unsigned char FieldBuf[FIELDBUF_SIZE];
#define FIELDBUF_SIZE   2100
```

Size of FieldBuf = (font's vertical size + 4) x text filed's horizontal size

Example: (font's vertical size (11) + 4) x text filed's horizontal size (140) = 2100

Scroll handling is defined as shown in the following code.
The following resource is defined within "include\gpcgui.h".

```
#define GUI_UP_SCROLL       (-1)
#define GUI_DOWN_SCROLL     1
#define GUI_MAX_SCROLL      10
```

Example)

```
guiScrollField((guiControl *)&control5, GUI_UP_SCROLL, SCROLL_LINES);
```

control5 is form data of the text field.
GUI_UP_SCROLL scrolls up.

```
#define SCROLL_LINES      1   // SCROLL LINES
```

SCROLL_LINES is line number for each scroll, and is defined in "main.h".

Data declaration for form and controls

It declares data used in the sample program.
For details on each structure, see the "5.4.1 Graphic User Interface" section in this manual.
The following example describes the "main.h" file located in the mouse sample directory "sample\gui_m\lf26".
Example) form declaration

```
const guiForm form1 = {0,0,160,240,
                       1,
                       6,
                       "Main Window",
                       (guiControl *)control1
                      };
```

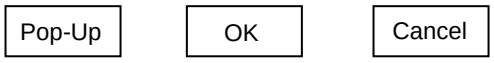
- The 1st to 4th items refer to the position and size of the button.
- The "1" in the 5th item is the form ID.
- The "6" in the 6th item is the number of controls in the form.
- The 7th item's "Main Window" is title of the form.
- The 8th item's control1 is a control used in the form.

Example) control button declaration

```
const guiControl control1[] = { 15, 20, 62, 18,
                                ID_POPUP,
                                GPC_BUTTON,
                                "Pop-Up",
                                (guiControlAttr *)0,
                                (guiControlGroup *)0,
                                (unsigned char *)0,
                                ;
                                ;
                               };
```

- The 1st to 4th item are position and size of button.
- The 5th item's ID_POPUP is the button's ID.
- The 6th item's GPC_BUTTON is the button's type.
- The 7th item's "Pop-Up" is the title of the button.
- The 8th item is the group attribute buffer. It is set to "0" here as it is not used.
- The 9th item is a group. It is set to "0" here as it is not used.
- The 10th item is the button's image. It is set to "0" here as it is not used.

The control button becomes as follows:



Example) image button declaration

Button's image declaration

```
const unsigned char up256[] = {
            0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f,
            0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f,,,
}
```

Image button declaration

```
const guiControl control5[] = { 10, 210, 40, 20,
                                ID_UP,
                                GPC_BUTTON,
                                "UP",
                                (guiControlAttr *)0,
                                (guiControlGroup *)0,
                                (unsigned char *)up256,
                                ;
                                ;
                               };
```

- The 1st to 4th item are position and size of button.
- The 5th item's ID_UP is the button's ID.
- The 6th item's GPC_BUTTON is the button's type.
- The 7th item's "UP" is the title of the button.
- The 8th item is the group attribute buffer. It is set to "0" here as it is not used.
- The 9th item is a group. It is set to "0" here as it is not used.
- The 10th item's up 256 is the button's image.

The image button becomes as follows:

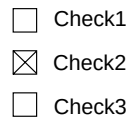
Example) check box declaration

This example sets the check box to ON when the button is pressed, and to OFF when it is pressed again. The control attribute's declaration applies to radio button and check box only.

```
guiControlAttr Ctrl2_Attr[6];
const guiControl control2[] = { 30, 40, 60, 20,
                                 ID_CHECK1,
                                 GPC_CHECK_BOX,
                                 "Check1",
                                 (guiControlAttr*)&Ctrl2_Attr[0],
                                 (guiControlGroup *)0,
                                 (unsigned char *)0,
                                 ;
                                 ;
                                };
```

- The 1st to 4th item are position and size of button.
- The 5th item's ID_CHECK1 is the button's ID.
- The 6th item's GPC_CHECK_BOX is the check box button's type.
- The 7th item's "Check1" is the title of the button.
- The 8th item's Ctr12_Attr[0] is the control's attribute.
- The 9th item is a group. It is set to "0" here as it is not used.
- The 10th item is the button's image. It is set to "0" here as it is not used.

The check box becomes as follows:



Example: radio button declaration

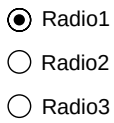
In this example, when there are more than one radio button, only one radio button is selected when the button is pressed.

The control attribute's declaration applies to radio button and check box only.

```
guiControlAttr Ctrl2_Attr[6];
const guiControl control2[] = { 30, 40, 60, 20,
                                ID_RADIO1,
                                GPC_RADIO_BUTTON,
                                "Radio1",
                                (guiControlAttr*)&Ctrl2_Attr[3],
                                (guiControlGroup*)groupRadio1,
                                (unsigned char *)0,
                                ;
                                ;
                               };
```

- The 1st to 4th item are position and size of button.
- The 5th item's ID_RADIO1 it the button's ID.
- The 6th item's GPC_RADIO_BUTTON is the radio button's type.
- The 7th item's "Radio1" is the title of the button.
- The 8th item's Ctrl2_Attr[3] is the control's attribute.
- The 9th item's groupRadio1 is a group.
- The 10th item is the button's image. It is set to "0" here as it is not used.

The radio button becomes as follows:



Example: group radio button declaration

For the same groupID, select only one from the group.

```
#define ID_RADIOGROUP1   150
```

The control attribute's declaration applies to radio button and check box only.

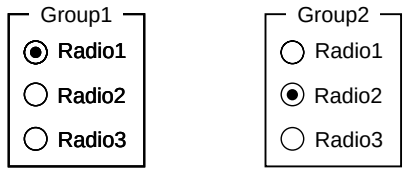
```
guiControlAttr Ctrl2_Attr[6];
const guiControlGroup groupRadio1[] = { 20, 120, 100, 60, ID_RADIOGROUP1,
                                        "Group1" };
```

- The 1st to 4th item are position and size of button.
- The 5th item's ID_RADIOGROUP1 is the group ID.
- The 6th item's "Group2" is the group title.

```
const guiControl control2[] = { 30, 170, 60, 20,
                                ID_RADIO3,
                                GPC_RADIO_BUTTON,
                                "Radio3",
                                (guiControlAttr*)&Ctrl2_Attr[5],
                                (guiControlGroup*)groupRadio1,
                                (unsigned char *)0,
                                ;
                                ;
                               };
```

- The 1st to 4th item are position and size of button.
- The 5th item's ID_RADIO3 is the button's ID.
- The 6th item's GPC_RADIO_BUTTON is the radio button's type.
- The 7th item's "Radio3" is the title of the button.
- The 8th item's Ctrl2_Attr[5] is the control's attribute.
- The 9th item's groupRadio1 is the radio group.
- The 10th item is the button's image. It is set to "0" here as it is not used.

The group radio buttons become as follows:



Program resource declaration

Control resource

Be careful so that there are no duplicate IDs of controls in the form when you declare control resources. Make sure also that there are no duplicate group IDs as well.

```
#define ID_POPUP          100  // POPUP form
#define ID_OPTION         101  // OPTION form
#define ID_DRAW           102  // DRAW form
#define ID_ERASE          103  // ERASE form
#define ID_ANIMATE        104  // ANIMATE form
#define ID_FIELD          105  // FIELD form

#define ID_OK             110  // OK button
#define ID_CANCEL         111  // CANCEL button
#define ID_UP             112  // UP button
#define ID_DOWN           113  // DOWN button

#define ID_CHECKGROUP1    120  // CHECK GROUP BOX 1
#define ID_CHECKGROUP2    130  // CHECK GROUP BOX 2
#define ID_CHECK1         121  // CHECK BOX1
#define ID_CHECK2         122  // CHECK BOX2
#define ID_CHECK3         123  // CHECK BOX3
#define ID_RADIOGROUP1    150  // RADIO GROUP button 1
#define ID_RADIOGROUP2    160  // RADIO GROUP button 2
#define ID_RADIO1         151  // RADIO button 1
#define ID_RADIO2         152  // RADIO button 2
#define ID_RADIO3         153  // RADIO button 3
```

5.5.3 Sample of Emulator Program on Window

This section explains how to create emulator program on Windows.

- This emulator program uses "gpcemu.c" in the "sample\gpc\src" directory.
- The main source of this sample program is "main.c" in the "sample\gpc\src" directory.
- The default size of the emulator screen is 640 × 480 dots.
- The emulator sample can use only 4BPP and 8BPP.

For Windows programming syntax, refer to the documentation of Windows SDK programming.

How to create emulator program on Windows

1. For the VC++5.0 environment file, prepare "gpc33emu.dsw" in the "sample\gpc\Lfemu" directory.

   Copy the object files needed for the emulator program ("gpcdraw.obj", "gpcdraw1.obj", "gpcdraw2.obj",

   "gpcdraw4.obj", "gpcdraw8.obj", "gpcgc.obj" and "gpcfont.obj") in the "sample\gpc\Lf26emu" directory to the same directory of "gpc33emu.dsw".

2. Prepare "gpcemu.c" in the "sample\gpc\ src" directory of the emulator program.

   Copy files required for the emulator program as well as "resource.h", "gpcemu.rc", "gpctoolbar.bmp" in the "sample\gpc\src" directory to the same directory of "gpcemu.c".

3. Prepare a main source file by referring to "main.c" in the "sample\gpc\src" directory.

   Add step execution processing by referring to 'How to create "main.c" of step execution processing'.

4. Open the VC++5.0 environment file "gpc33emu.dsw", delete the files in the "Source Files" folder, then add the programs files for use in the "Source Files" folder.

   As the Include directory common for graphics libraries is now set as [/I "..\ ..\ ..\ include"] by Project Options, set it according to the sample program.

   For details on how to add files, refer to the Microsoft VC++5.0 manual.

Step execution

The step execution handling in the sample executes steps for checking the graphic library.

- When run on DMT33006, the step processing is done automatically.
- When run using the emulator, the step processing is done by using toolbar button.

Sample for executing steps

First, declare the global variable gpcCount. Then create and use it as shown in the following code when step execution on GPC_STEP_PROCESS is desired.

```
#if defined(_WINDOWS)
    // The gpcStepProcess( ) function is set up at "src\gpcemu.c"
    extern int  gpcStepProcess(void);
    #define  GPC_STEP_PROCESS  if(gpcStepProcess( )){return 0;}
#else
    #define  GPC_STEP_PROCESS  (delay(3000))
#endif

int gpcCount;
```

Example)

Set in the SampleStart( ) function

```
int SampleStart( )
{
    // Initialize step execution
    gpcCount = 0;
    // Step Processing
    GPC_STEP_PROCESS;

    gpcSetColor(GPC_LIGHTGREEN);
    gpcDrawEllipse(10, 170, 80, 220);

    // Step Processing
    GPC_STEP_PROCESS;
    ;
}
```

Emulater declaration

The following declaration is defined in "gpcemu.c".

Declaration of define

- The color table is set as shown in the following code.

```
#if LF26
#define GPC_COLOR_TABLE gpcSetColorTable((unsigned char *)&gpcCOLOR8, 256);
#elif LF37
#define GPC_COLOR_TABLE gpcSetColorTable((unsigned char *)&gpcCOLOR4, 16);
#endif
```

- Palette table is set as shown in the following code.

```
#if LF26
#define GPC_PALETTE_ENTRY gpcSetPaletteEntry((unsigned char *)&gpcCOLOR8,256);
#elif LF37
#define GPC_PALETTE_ENTRY gpcSetPaletteEntry((unsigned char *)&gpcCOLOR4, 16);
#endif
```

When using a new color table, call GPC_PALETTE_ENTRY and then call GPC_COLOR_TABLE.
For the example, see the explanation on the gpcPaintBitMap( ) function described later in this manual.

Declaration of external variables

```
extern unsigned char gpcCOLOR4[];      // 4 bits/pixel color table
extern unsigned char gpcCOLOR8[];      // 8 bits/pixel color table
extern int gpcWidth;                   // Horizontal size of screen
extern int gpcHeight;                  // Vertical size of screen
extern int gpcColorBPP;                // Color bit/pixel
```

Declaration of step execution

```
// gpcCount is defined in "src\main.c"
extern int gpcCount;
// Global declaration
int gpcStep;

int gpcStepProcess(void)
{
    if(gpcStep == gpcCount++){
        return 1;
    }
    return 0;
}
```

Emulator program on Windows

The following source is defined in "gpcemu.c".

Example) window message handling function

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM
lParam )
{
UINT  ButtonID;
int iLoop;
switch( uMsg )
{
    case WM_CREATE :
          // Initialize window
          // Create toolbar
          InitCommonControls( );
          gpcToolBarCreate(hWnd);

          // Set up palette table for use by program
          GPC_PALETTE_ENTRY

          // Initialize variable for step execution
          gpcStep=0;

          // Coll main( ) function
          gpcPaintBitMap(hWnd);

          // Send message to display initial screen
          LOWORD(wParam) = IDM_FIRST;
          PostMessage(hWnd, WM_COMMAND, wParam, lParam);

          // Set according to graphic handling screen
          SetWindowPos(hWnd, HWND_TOP, 0, 0, gpcWidth + 10,
          gpcHeight + 60, SWP_NOMOVE );
          break;
      case WM_PAINT :
          // Set here in order to display screen again
          {
          PAINTSTRUCT ps;
          HDC hMemDC;

          BeginPaint( hWnd, &ps );
          hMemDC = CreateCompatibleDC( ps.hdc );

          // Set up again according to color palette set at program
          SelectPalette (hMemDC, hPal, FALSE);
          RealizePalette (hMemDC);

          SelectObject( hMemDC, hBitmap );
          BitBlt( ps.hdc, 0, TOOLBAR_SIZE,  gpcWidth+1,
          gpcHeight+1,hMemDC, 0, 0, SRCCOPY );
          break;
          }
      case WM_COMMAND :
          switch( LOWORD( wParam ) )
          {
              case ID_STEP :
              case IDM_STEP :
                  // Handling for when toolbar button for step execution is pressed
                  // Increment global variable for step execution
```

```
                    gpcStep++;
                    // Call main( ) function for step execution
                    gpcPaintBitMap(hWnd);

                    //  Display screen again
                    RedrawWindow(hWnd, NULL, NULL, RDW_INVALIDATE);
                    break;
                case ID_FIRST :
                case IDM_FIRST :
                    // Handling for when start's  toolbar button is pressed
                    // Initialize global variables for step execution
                    gpcStep = 0;

                    // Call main( ) function
                    gpcPaintBitMap(hWnd);

                    // Display screen again
                    RedrawWindow(hWnd, NULL, NULL, RDW_INVALIDATE);
                    break;
            }
            break;
        default :
            return( DefWindowProc( hWnd, uMsg, wParam, lParam ) );
    }
    return( 0L );
}
```

Example) explanation of the gpcPaintBitMap( ) function for displaying in the emulator screen

```
void APIENTRY gpcPaintBitMap(HWND  hWnd)
{
    HDC   hDC;
    // Declare display memory's external variables
    extern unsigned char gpcDispMem[];

    // Set up bitmap
    gpcSetBitMap( );
    hDC = GetDC( hWnd );
    lpbi = (BITMAPINFOHEADER*)Temp;
    *lpbi = bi;

    // Set up palette table for use by program
    GPC_PALETTE_ENTRY

    // Set up color palette table for use by program
    GPC_COLOR_TABLE

    // Call main( ) function
    main( );

    // Set up again according to color palette set at program
    SelectPalette (hDC, hPal, FALSE);
    RealizePalette (hDC);
    hBitmap = CreateDIBitmap( hDC, &bi, CBM_INIT, gpcDispMem,
    (LPBITMAPINFO)lpbi, DIB_RGB_COLORS);

    // Release device
    ReleaseDC( hWnd, hDC );
    return ;
}
```

5.6 Precautions

- If the pop-up window is to be used, be sure to use gpcSetMemoryBuf( ) to define memory and prepare sufficient buffer size.
- Be sure to set only one general button for the control of the pop-up window.
- If the text field is to be used, be sure to use guiSetFieldText( ) to prepare sufficient buffer size.
- If tasks are to be used, use the vcre_tsk( ) function in main( ) to define all of the tasks to execute. Operation of a system call using an ID not defined is not guaranteed.
- For idle tasks, set acceptability and priority to the lowest level. Also, do not return from the idle task.
- Adding GUI to GRAPHIC33 requires ROS33, which can be purchased separately.

Appendix Verifying Operation with DMT33 Boards

This section describes how to verify the operation of graphic processing by executing a sample program using E0C33 Family demonstration tools, the DMT33006, DMT33MON, and DMT33LCD.

For description of the system configuration and connection method ,refer to the DMT33006 and DMT33LCD manuals.

A.1 System Setting

DMT33006 setting

Set the DIP switches and jumper switches of DMT33006 as follows:
The settings are all default values. If the current settings are the same as the factory settings when the board was shipped, then you do not need to change them.

- Set the LCD panel power supply voltage to 5V. (1-2 short of JP2)
- Set the VDDE2 (voltage for LCD interface) to 5V. (1-2 short of JP4)
- Set the OSC3(20MHz)to 1/3 cycle for the clock exclusive for the controller of the built- in LCD (CLKT). (Set DSW1's SW6 to OFF, SW7 to OFF, and SW8 to ON)

DMT33006 connections

Connect the DMT33006 + DMT33MON to the PC.
Set the DMT33MON's DEBUG SW to ON.

When running the FL26 sample:
Connect the J3 connector of DMT33006 and the J1 connector of DMT33LCD26 using the supplied flat cable. Connect so the red line on the flat cable appears on the REC switch side of DMT33006.
DMT33LCD26 is not needed to be connect to the power source separately, as the +5V power is supplied from the J1 connector.

When running the LF37 sample:
Connect the J3 connector of DMT33006 and the J1 connector of DMT33LCD37 using the supplied flat cable. Connect so the red line on the flat cable appears on the REC switch side of DMT33006.
DMT33LCD37 is not needed to be connect to the power source separately, as the +5V power is supplied from the J1 connector.

Precaution concerning the system

- Be sure to turn off all power sources on the system before installing or removing the other board on DMT33LCD.
- The area on the VRAM (0x380000 -) of E0C332L01 shifts by 1 line when compared with the LCD display area. Therefore, the second line on VRAM appears on the first line on the LCD.
- The last line (240th line) on VRAM of E0C332L01 appears on the 239th line on the LCD.
- Be careful when handling the LCD panel and the backlight as the glue d surface between them is weak.

Note: Before disconnecting boards or apparatus from the system, be sure to turn off all of the power of the boards or apparatus. For information on handling the boards, refer to the "E0C33 Family Demonstration Board Manual."

A.1.1 Software

The PC hosting the GRAPHIC33 must have the "E0C33 Family C Compiler Package" development tools installed on it.
To download your program into the DMT33006 using the debug monitor, you must have debugger (db33) ver. 1.72 or later.

A.2 Sample Program Execution Procedure

A.2.1 Preparatory Operations before Startup

Since each sample program directory "gpclib\sample\" contains absolute object files in executable format, there is no need to compile or link the sample program specifically. Also batch files from which you can launch the debugger are provided. The following explains how to start each sample program after downloading them to the DMT33006.

∗ Be sure to reset the reset(SW5)key of DMT33006 before running each sample.

Running the graphics function sample

```
\gpc
├── \lf26
│   └ run_8bpp.bat      8BPP sample executable batch program
└── \lf37
    └ run_4bpp.bat      4BPP sample executable batch program
```

When running the graphics function sample:
Move to the directory of each sample.
Reset the reset (SW5) key of DMT33006, then start run_xbpp.bat in the directory of each sample in the DOS prompt.

Example)DOS>run_8bpp.bat // for lf26
Example)DOS>run_4bpp.bat // for lf37

∗ This sample displays the drawing and text automatically.

Running the GUI Key Event Sample

```
\gpc_k
├── \lf26
│   └ run_8bpp.bat      8BPP sample executable batch program
└── \lf37
    └ run_4bpp.bat      4BPP sample executable batch program
```

When running the GUI key event sample:
Move to the directory of the each sample.
Reset the reset (SW5) key of DMT33006, then start run_xbpp.bat in the directory of each sample in the DOS prompt.

Example) DOS>run_8bpp.bat // for lf26
Example) DOS>run_4bpp.bat // for lf37

∗ For key, move by pressing between key 50 and key 54 (SW7) of the DMT33006 board.
Use K50(↑), K51(←), K52(↓), K53(→), K54(Enter)
If you press the ↑ (Key), ← (Key), ↓ (Key), → (Key) key at the current position, the cursor position moves.
In the case of pressing a button: pressing Enter(Key) reverses the display.
In the case of releasing a button: releasing Enter(Key) reverses the display and processes the button.

Running GUI Mouse Event Sample

```
\gpc_m
├── \lf26
│   └ run_8bpp.bat      8BPP sample executable batch program
└── \lf37
    └ run_4bpp.bat      4BPP sample executable batch program
```

When running the GUI mouse event sample:
Move to the directory of each sample.
Reset the reset(SW5) key of DMT33006, then start run_xbpp.bat in the directory of each sample in the DOS prompt.

Example) DOS>run_8bpp.bat //for lf26
Example) DOS>run_4bpp.bat //for lf37

∗ For mouse, move by pressing between key 50 and key 54 (SW7) of the DMT33006 board. Use K50(↑), K51(←), K52(↓), K53(→), K54(Enter).
If you press the ↑ (mouse), ← (mouse), ↓ (mouse), → (mouse) key at the current position of the mouse, the mouse position moves.
In the case of pressing a button: pressing Enter(Key) reverses the display.
In the case of releasing a button: releasing Enter(Key) reverses the display and processes the button.

Palette Sample Execution

```
\gpcpal
└──run_pal.bat      Palette sample executable batch program
```

When running the palette sample:
Move to the directory of each sample.
Reset the reset(SW5) key of DMT33006, then start run_xbpp.bat in the directory of each sample in the DOS prompt.
Example) DOS> run_pal.bat
∗ This sample displays the palette image automatically.

Running Emulator Sample

```
\gpc
├── \lf26emu
│   └ gpc33emu.exe     Executable file for lf26 of Window Emulator
└── \lf37emu
    └ gpc33emu.exe     Executable file for lf37 of Window Emulator
```

When running the Emulator sample:
Move to the directory of each sample.
From the Windows Explorer, double-click and run gpc33emu.exe in the directory.

∗ Running the sample Step displays drawing and text.
Toolbar(→) displays drawing and text step by step.
Toolbar(First) returns to the start position for viewing from the beginning.

A.3 Making a Program

The sample programs can be changed before testing, as necessary. For reference, the *make* procedure is explained below.

When you modify the sample, modify and then rebuild the program, then refer to the program execution procedure to run the sample program.

Graphic Function Sample Build

```
\gpc
├── \lf26
│   └ make_8bpp.bat     8BPP sample Make batch program
├── \lf37
│   └ make_4bpp.bat     4BPP sample Make batch program
└──\src                 Common source directory
```

The source directory(+--\ src) is shared among lf26, lf37, lf26emu, and lf37emu.

When building the graphics function sample:

Move to the environment directory of each sample.

Start make_xbpp.bat in the environment directory of each sample in the DOS prompt to build.

Example) DOS>make_8bpp.bat // for lf26

Example) DOS>make_4bpp.bat // for lf37

GUI Key Event Sample Build

```
\gpc_k
├── \lf26
│   └ make_8bpp.bat     8BPP sample Make batch program
├── \lf37
│   └ make_4bpp.bat     4BPP sample Make batch program
└──\src                 Common source directory
```

The source directory(\ src) is shared between lf26 and lf37.

When building the GUI key event sample:

Move to the environment directory of each sample.

Start make_xbpp.bat in the environment directory of each sample in the DOS prompt to build.

Example) DOS>make_8bpp.bat // for lf26

Example) DOS>make_4bbp.bat // for lf37

GUI Mouse Event Sample Build

```
\gpc_m
├── \lf26
│   └ make_8bpp.bat     8BPP sample Make batch program
└──\lf37
    └ make_4bpp.bat     4BPP sample Make batch program
```

When building the GUI mouse sample:

Move to the environment directory of each sample.

Start make_xbpp.bat in the environment directory of each sample in the DOS prompt to build.

Example) DOS>make_8bpp.bat // for lf26

Example) DOS>make_4bbp.bat // for lf37

Palette Sample Build

```
\gpcpal
└── make_pal.bat      Palette sample Make batch program
```

When building the Palette sample:
Start make_pal.bat in the directory in the DOS prompt to build.
Example) DOS> make_pal.bat

Emulator Sample Build

```
\gpc
├── \lf26emu
│   └ gpc33emu.dsw      Environment build file of VisualC++
├── \lf37emu
│   └ gpc33emu.dsw      Environment build file of VisualC++
└──\src                 Common source directory
```

The source directory (\src) is shared among lf26, lf37, lf26emu, and lf37emu.
Open each project work space gpc33emu.dsw under the \gpc directory with Visual C/C++5.0 to build.
For 8BPP, use gpc33emu.dsw in the \ lf26emu directory.
For 4BPP, use gpc33emu.dsw in the \ lf37emu directory.
∗ Do not delete the object files gpcdraw.obj, gpcdraw1.obj, gpcdraw2.obj, gpcdraw4.obj, gpcdraw8.obj, gpcgc.obj, gpcfont.obj.

Make can be executed from Workbench wb33. (Refer the "E0C33 Family C Compiler Package Manual" for details)

EPSON International Sales Operations

AMERICA

EPSON ELECTRONICS AMERICA, INC.

- HEADQUARTERS -
1960 E. Grand Avenue
El Segundo, CA 90245, U.S.A.
Phone: +1-310-955-5300 Fax: +1-310-955-5400

- SALES OFFICES -

West
150 River Oaks Parkway
San Jose, CA 95134, U.S.A.
Phone: +1-408-922-0200 Fax: +1-408-922-0238

Central
101 Virginia Street, Suite 290
Crystal Lake, IL 60014, U.S.A.
Phone: +1-815-455-7630 Fax: +1-815-455-7633

Northeast
301 Edgewater Place, Suite 120
Wakefield, MA 01880, U.S.A.
Phone: +1-781-246-3600 Fax: +1-781-246-5443

Southeast
3010 Royal Blvd. South, Suite 170
Alpharetta, GA 30005, U.S.A.
Phone: +1-877-EEA-0020 Fax: +1-770-777-2637

EUROPE

EPSON EUROPE ELECTRONICS GmbH

- HEADQUARTERS -
Riesstrasse 15
80992 Munich, GERMANY
Phone: +49-(0)89-14005-0 Fax: +49-(0)89-14005-110

SALES OFFICE
Altstadtstrasse 176
51379 Leverkusen, GERMANY
Phone: +49-(0)2171-5045-0 Fax: +49-(0)2171-5045-10

UK BRANCH OFFICE
Unit 2.4, Doncastle House, Doncastle Road
Bracknell, Berkshire RG12 8PE, ENGLAND
Phone: +44-(0)1344-381700 Fax: +44-(0)1344-381701

FRENCH BRANCH OFFICE
1 Avenue de l' Atlantique, LP 915 Les Conquerants
Z.A. de Courtaboeuf 2, F-91976 Les Ulis Cedex, FRANCE
Phone: +33-(0)1-64862350 Fax: +33-(0)1-64862355

ASIA

EPSON (CHINA) CO., LTD.
28F, Beijing Silver Tower 2# North RD DongSanHuan
ChaoYang District, Beijing, CHINA
Phone: 64106655 Fax: 64107319

SHANGHAI BRANCH
4F, Bldg., 27, No. 69, Gui Jing Road
Caohejing, Shanghai, CHINA
Phone: 21-6485-5552 Fax: 21-6485-0775

EPSON HONG KONG LTD.
20/F., Harbour Centre, 25 Harbour Road
Wanchai, Hong Kong
Phone: +852-2585-4600 Fax: +852-2827-4346
Telex: 65542 EPSCO HX

EPSON TAIWAN TECHNOLOGY & TRADING LTD.
10F, No. 287, Nanking East Road, Sec. 3
Taipei
Phone: 02-2717-7360 Fax: 02-2712-9164
Telex: 24444 EPSONTB

HSINCHU OFFICE
13F-3, No. 295, Kuang-Fu Road, Sec. 2
HsinChu 300
Phone: 03-573-9900 Fax: 03-573-9169

EPSON SINGAPORE PTE., LTD.
No. 1 Temasek Avenue, #36-00
Millenia Tower, SINGAPORE 039192
Phone: +65-337-7911 Fax: +65-334-2716

SEIKO EPSON CORPORATION KOREA OFFICE
50F, KLI 63 Bldg., 60 Yoido-dong
Youngdeungpo-Ku, Seoul, 150-763, KOREA
Phone: 02-784-6027 Fax: 02-767-3677

SEIKO EPSON CORPORATION
ELECTRONIC DEVICES MARKETING DIVISION

Electronic Device Marketing Department
IC Marketing & Engineering Group
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5816 Fax: +81-(0)42-587-5624

ED International Marketing Department Europe & U.S.A.
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5812 Fax: +81-(0)42-587-5564

ED International Marketing Department Asia
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-(0)42-587-5814 Fax: +81-(0)42-587-5110

In pursuit of **“Saving” Technology**, Epson electronic devices.
Our lineup of semiconductors, liquid crystal displays and quartz devices
assists in creating the products of our customers’ dreams.
Epson IS energy savings.

EPSON

SEIKO EPSON CORPORATION
ELECTRONIC DEVICES MARKETING DIVISION

■ EPSON Electronic Devices website

http://www.epson.co.jp/device/

Issue NOVEMBER 2000, Printed in Japan ⓦ A