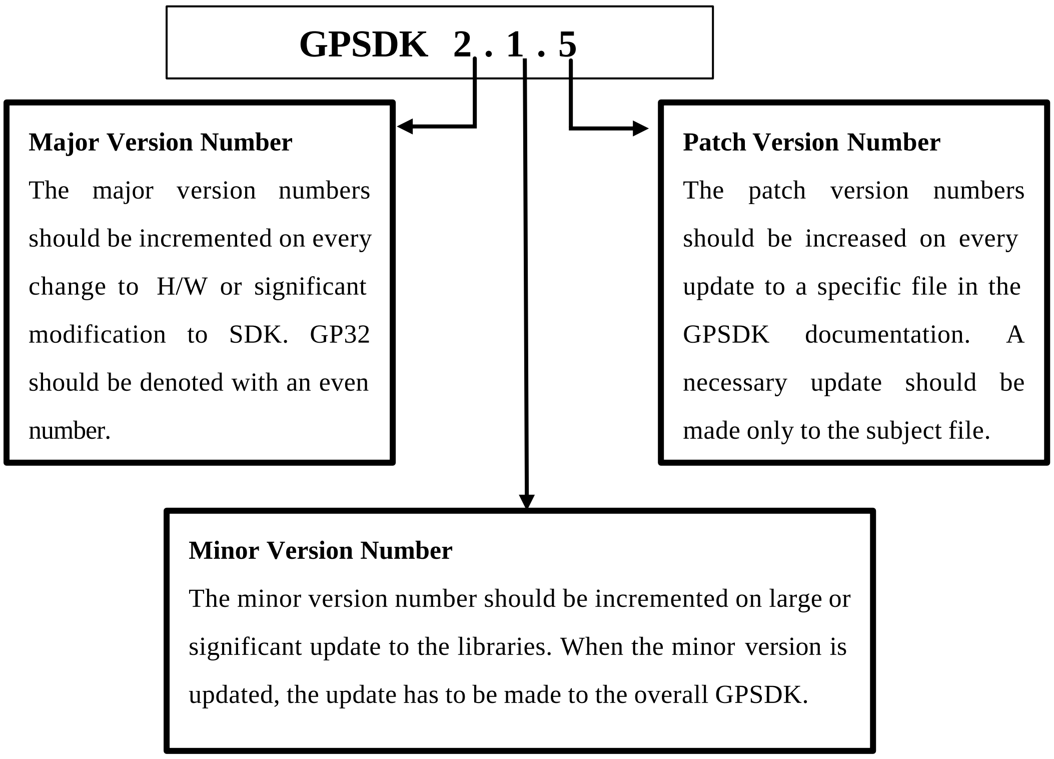
(GP32 Basic Manual for ARM SDT Developer)

Version 2.1.5

April, 1, 2002
Version: 2.1.5
Developed by Achi Jung & Ji-Hyeon Lim

Table of Contents

1. Version Overview



GPSDK 2 . 1 . 5

Major Version Number
The major version numbers should be incremented on every change to H/W or significant modification to SDK. GP32 should be denoted with an even number.

Patch Version Number
The patch version numbers should be increased on every update to a specific file in the GPSDK documentation. A necessary update should be made only to the subject file.

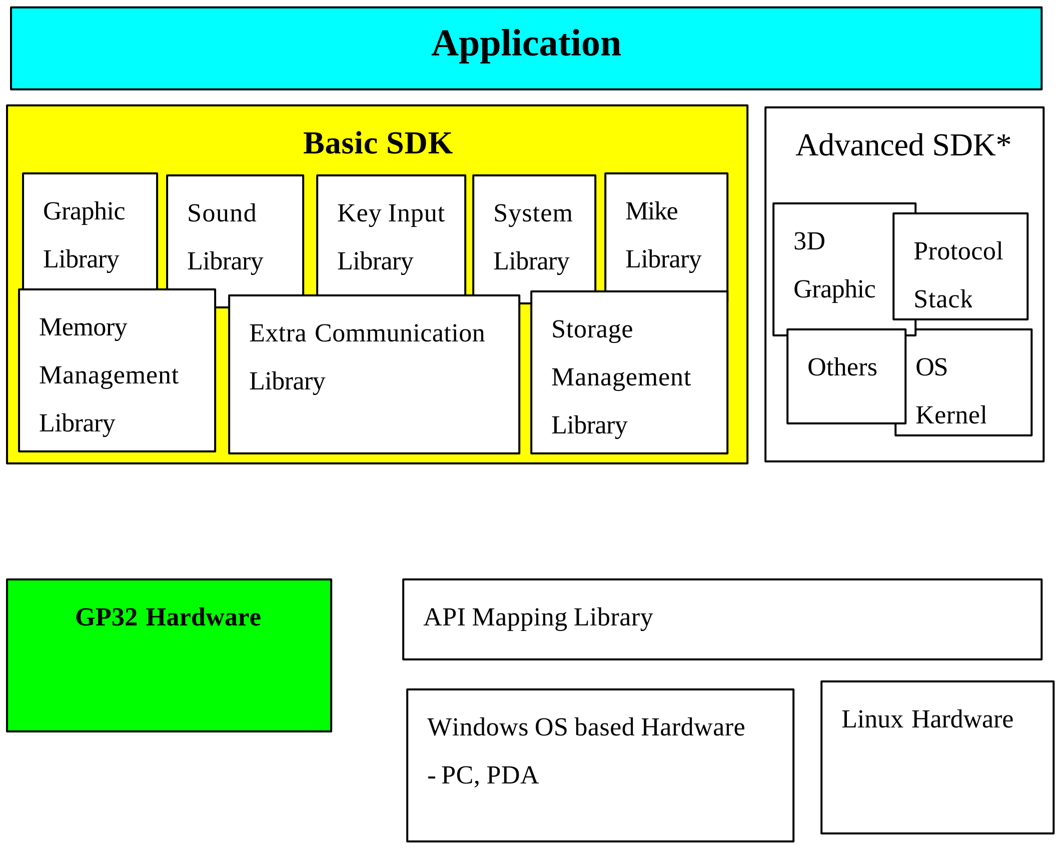
Minor Version Number
The minor version number should be incremented on large or significant update to the libraries. When the minor version is updated, the update has to be made to the overall GPSDK.

Version History

2001.1	: GPSDK Version 1.0 release	(ARM7TDMI CPU)
2001.4	: GPSDK Version 1.1 release	
2001.10	: GPSDK Version 2.0.0 release	(ARM920T CPU)
2001.11	: GPSDK Version 2.0.1 release	
2001.12	: GPSDK Version 2.1.0 release	
2002.4	: GPSDK Version 2.1.5 release	

2. SDK Overview

The following diagram explains you how GP32 Software development module is configured.



<Figure 1> SDK Configuration

GP32 software development module consists of Basic SDK(Software Development Kit) and Advanced SDK. In addition, an API Mapping Library is provided to port to other hardware environments, enabling GP32 to maintain compatibility with respect to GP32 dedicated software source code and resource level.

The basic GP32 SDK comprises API(Application Program Interface) Library functions for graphics, sound, key input, microphone, system, memory management, extra communications and storage management. These are basic and powerful libraries required to produce programs easily on GP32 hardware.

The advanced SDK provides real-time multi-tasking operating system and related communication protocol stack, 3D graphic library, and other advanced libraries. These are useful in developing stronger and more sophisticated applied software.

Furthermore, AML(API Mapping Library) provides a convenient and powerful solution to port GP32 dedicated applied programs made by using the 2 SDKs mentioned above to a different hardware environment with respect to the source.
This manual details the basic knowledge needed to develop game application programmed for GP32 hardware based on the Basic SDK.

The following is an overview of GP32 system performance.

The GP32 graphic system has a resolution of 320x240 and also supports 8bit Palette Mode and 16bit True Color Mode.

This is the highest resolution ever generated for handheld game systems. Considering the future resolution(640x480) for handhelds, it is the main factor making porting easier and facilitating porting to diverse hardware.

Regarding sound, MIDI is supported for background sound providing high sound quality with low data volume. Thus, this system scores high among games. PCM sound is also provided which supports 16bit sound in diverse sampling data(11.025kHz, 22.5kHz, 44kHz) on stereo. It can mix and output up to 4 channels simultaneously on real-time.

The main features of GP32 game system is as follows:

1. **320x240 resolution, 65,536 concurrent colors**
2. **8bit and 16bit PCM stereo sound, 4 channels mixing and output in real-time**

Another feature is SMC(Smart Media Card) that makes large-volume storage management possible. As it supports DOS file system, large-volume games can be produced linked with 8MB high-speed internal SDRAM.

3. **SMC(Smart Media Card) – 16MB is basic. 32MB and 64MB are possible.**
4. **File system linking SMC with 8MB high-speed internal SDRAM, storage allocation function**

C language is also supported, which makes program development much easier.

5. **C language development**

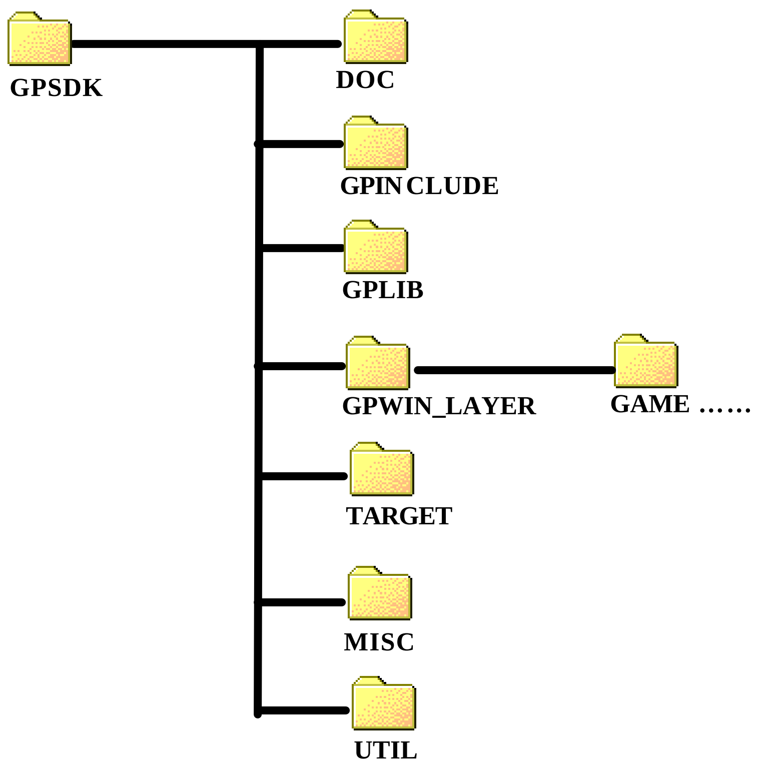
C language is the basis of development, so developers must be informed.

3. GP32 Development Environment

The following seven items comprising the development environment shall be provided for a GP32 application software developer.

No.	Item	Definition
1	GP32	Game Device
2	GPSDK	Development Software Library and API Layer for VC++
3	Multi-ICE or MAJIC	Hardware to load software already developed in PC environment to GP32 (Option)
4	Interface board	Connecting board between Multi-ICE or MAJIC and GP32 (Option)
5	ARM SDT 2.5 Tools Or ADS 1.2	Development software operating on the PC such as ARM Assembler, C/C++ Compiler and Linker
6	ADW	ARM Debugger for Windows (Debugging software operating on PC) (Option)
7	GP32 Dev Utility	Application program, running on Windows, which provides various functionalities including data conversion, transmission, SMC read and write and others.

The basic version 2.1.5 of GP32 SDK has been illustrated as shown below for your clear understanding.



<Figure 2-1> GPSDK_BV20 Folder

GPSDK \ DOC

This folder contains guidelines and references designed to help a developer creating application programs using GPSDK.

GPSDK \ GPINCLUDE

This folder includes GPSDK API header files. The headers should be included when GP32 is targeted.

GPSDK \ GPLIB

GPSDK API libraries contain ALF files and Object files with the .o extension. These files can be imported for use when GP32 is targeted.

GPSDK \ GPWIN_LAYER

This folder includes files required for you to develop application programs in Visual C++ tool based on APM layer.

GPSDK \ GPWIN_LAYER \ GAME

This folder is designed to contain the actual application source codes.

GPSDK \ GPWIN_LAYER \ GAME \ EXAMPLES

This folder includes all the programming examples for both VCGP32 and VC++.

GPSDK \ GPWIN_LAYER \ GPINCLUDE_WIN

This folder contains completely separate header files for VC++ environment.

GPSDK \ GPWIN_LAYER \ GPLIB_WIN

This folder includes library files for VC++ environment.

GPSDK \ GPWIN_LAYER \ VIRTUAL_SMC

This folder saves the virtual SMC files created for interface compatibility with SMC, the external storage medium for GP32 when you develop applications in VC++ based environment.

GPSDK \ TARGET

This folder contains files necessary to control the options relevant to environment setting when

GP32 is targeted.

GPSDK \ MISC

This folder includes other useful sources to develop applications such as streaming wave out and GP32 character input.

GPSDK \ UTIL

This folder contains a variety of utility and Config files required to develop applications on GP32.

3.1 Development Environment Configuration

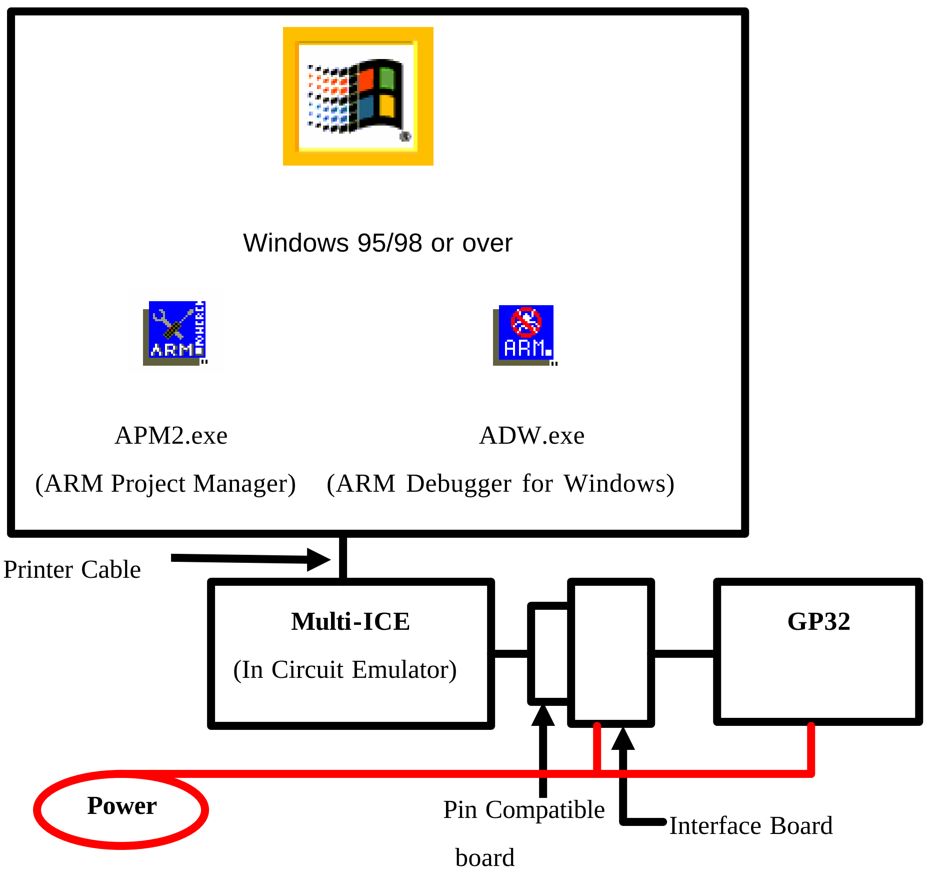
GP32 ARM SDT developer should support the implementation of the following environment.

1. Windows 95/98/ME or Windows NT Workstation 4.0 or over
2. ARM SDT 2.5 or over
3. Multi-ICE or MAJIC (In Circuit Emulator)
4. Over 10Mbps of LAN environment (or Serial / Parallel Connection)

1) When MAJIC is used as an in circuit emulator,

Further details to follow soon.

2) When Multi-ICE is used as an emulator,



<Figure 2-2-1> Development Environment Configuration

Once you install ARM SDT 2.5 in Windows, each executable program for ARM Project Manager and ARM Debugger for Windows is generated as shown in the figure 2-2-1. Also, Multi-ICE should be connected to PC using Printer Cable. Notably, it is much convenient to separate the power of JEENI and GP32 from the existing power to connect to another power source like multi tap. You shall install Multi-ICE Serve that comes with Multi-ICE and copy 2400.cfg provided by GamePark to Multi-ICE folder. Then, please follow below instructions to initialize Multi-ICE Server. For further details, please refer to the Multi-ICE User's Guide.

To summarize,

1. Install ARM SDT 2.5.
2. Install Multi-ICE Server.
3. Copy 2400.cfg to Multi-ICE folder.
4. Connect Multi-ICE to PC.
5. Execute and initialize Multi-ICE Server:
    - i) Select the Start-up Option from the Settings menu bar,
    - ii) Go to the Start-up Configuration in the Start-up Option,
    - iii) Click on the Load Configuration
    - iv) Load 2400.cfg from the Multi-ICE folder listed under the Loaded File

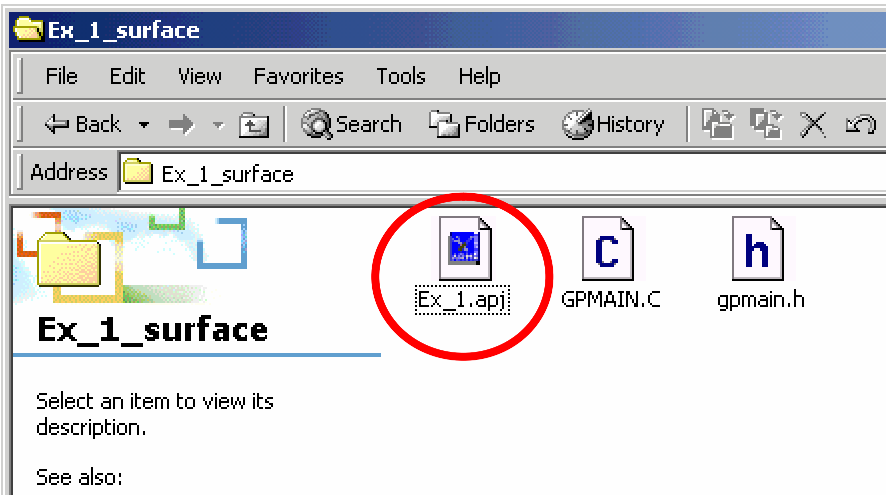
3.2 Starting APM

All the apj files fully explained below are located in the directory GPSDK \ GPWIN_LAYER \ GAME \ EXAMPLES.

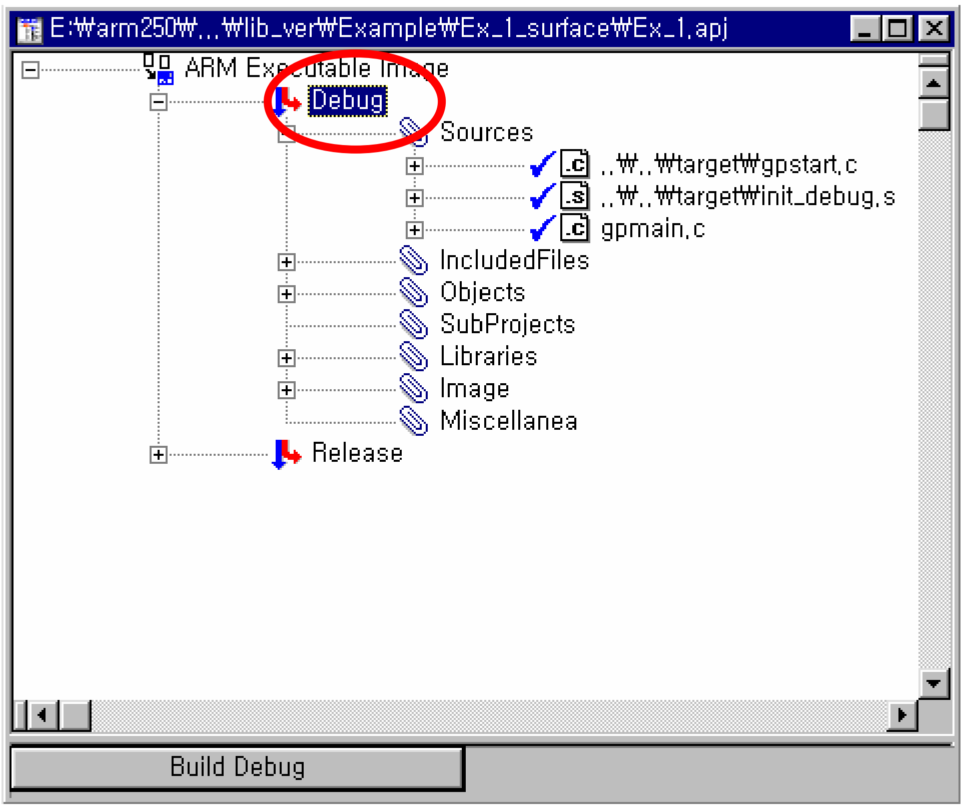
In the EXAMPLE folder of Basic GP32 SDK, you will find a folder named Ex_1_Sur and 3 files inside it.

<Figure 2-3>

1. Ex_1.apj
2. Gpmain.c
3. Gpmain.h

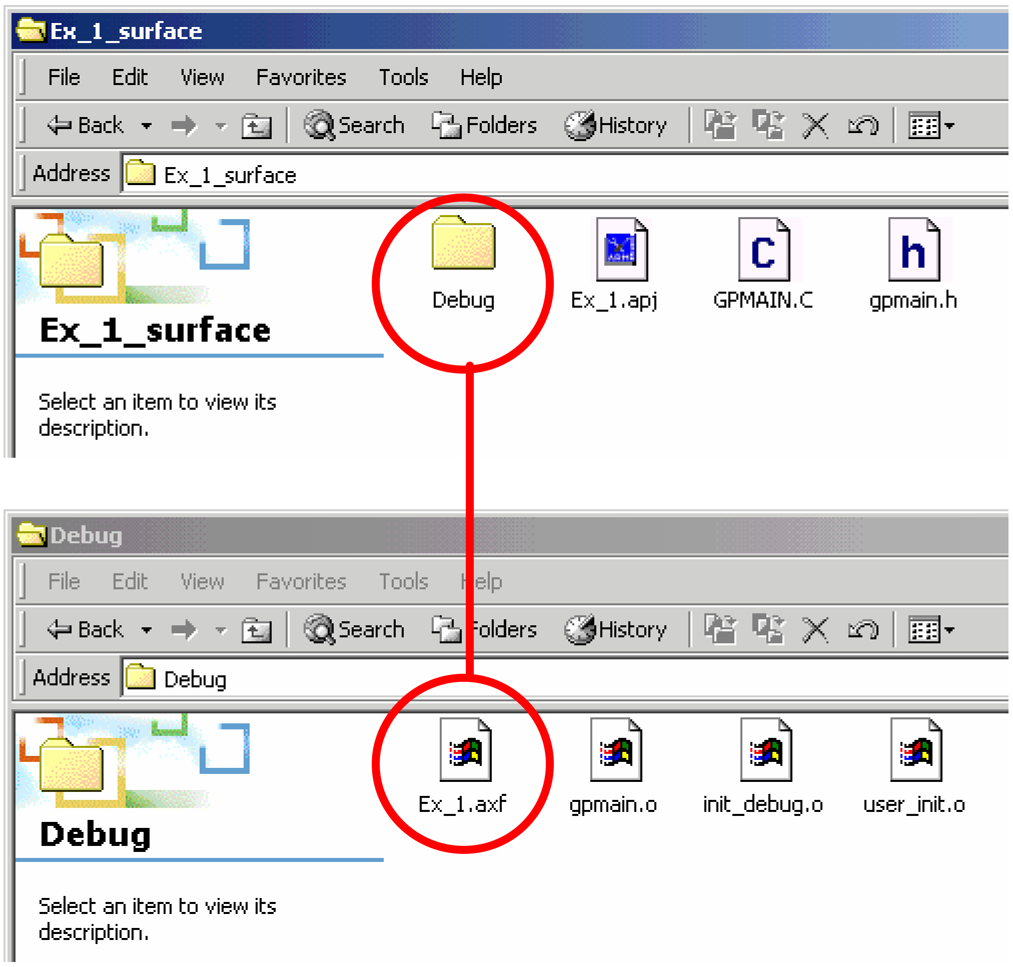

Ex_1_surface
File Edit View Favorites Tools Help
Back Search Folders History
Address Ex_1_surface
Ex_1_surface
Ex_1.apj
GPMAIN.C
gpmain.h
Select an item to view its description.
See also:

Double click on Ex_1.apj to run ARM Project Manager. The project window of APM will be as shown below (if ARM SDT 2.5 is installed).



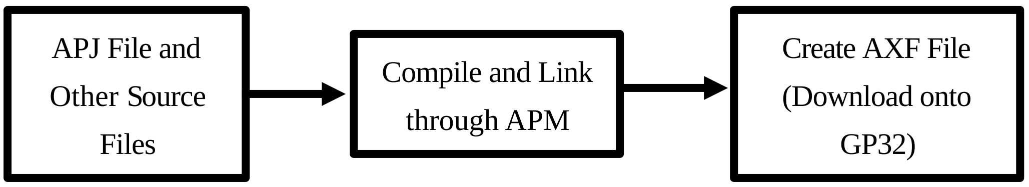
<Figure 2-4> APM Project Window

Here, select Debug from the Tree Item and Build Debug will show on the button at the bottom of the window. Click on the Build Debug button in order to compile and link.
As shown below, a Debug folder will be created in the Ex_1_Sur folder and 3 object files and an Ex_1.axf file will be created in the Debug folder.


Ex_1_surface
File Edit View Favorites Tools Help
Back Search Folders History
Address Ex_1_surface
Ex_1_surface
Select an item to view its description.
Debug
Ex_1.apj
GPMAIN.C
gpmain.h
Debug
File Edit View Favorites Tools Help
Back Search Folders History
Address Debug
Debug
Select an item to view its description.
Ex_1.axf
gpmain.o
init_debug.o
user_init.o

<Figure 2-5> Newly Created Debug Folder and AXF File

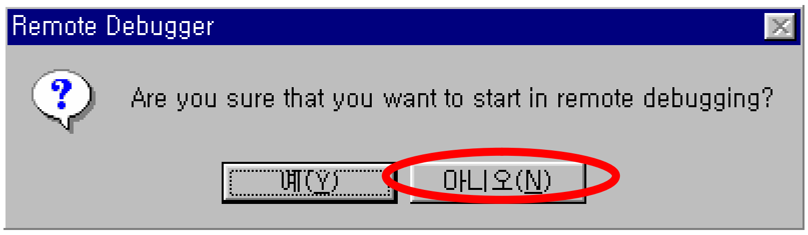
The following diagram shows how to download the newly created Ex_1.axf file onto GP3 through Multi-ICE using ADW(ARM Debugger for Windows) program.


APJ File and Other Source Files
Compile and Link through APM
Create AXF File (Download onto GP32)

<Figure 2-6> Creating AXF File through APM

3.3 ADW/Multi-ICE Environment Configurations

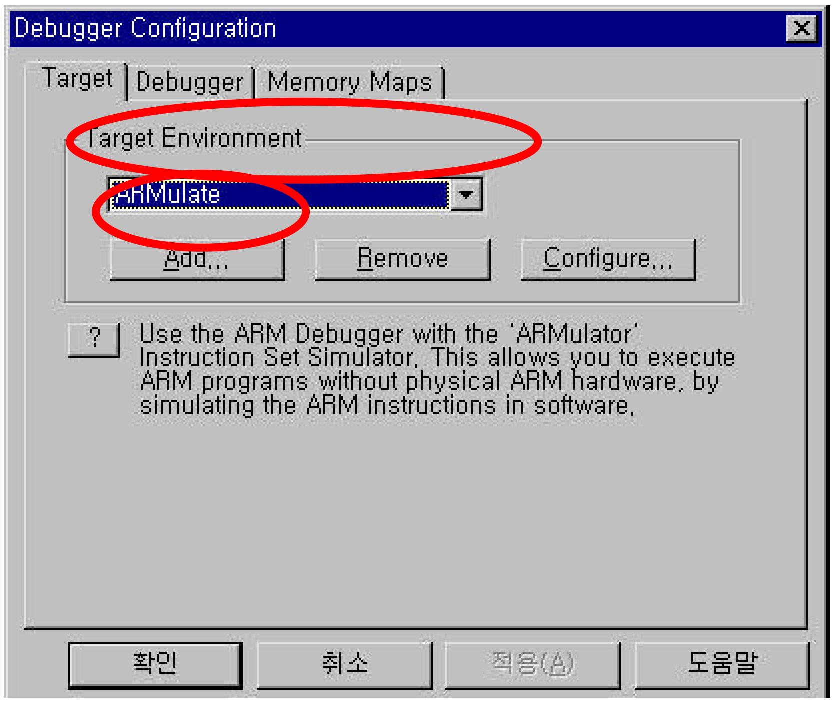
Configuration for ADW to recognize Multi-ICE is required. How to configure is explained with an example of using Multi-ICE.
Run ADW and the following message window will appear.


Remote Debugger

Are you sure that you want to start in remote debugging?

<Figure 2-7> Message Window when Running ADW (Select "No")

First, click on "No" in the above dialog box. Then, from the ADW menu bar, go to Options->Configure **Debugger** to bring up the **Debugger Configuration box**.


Debugger Configuration

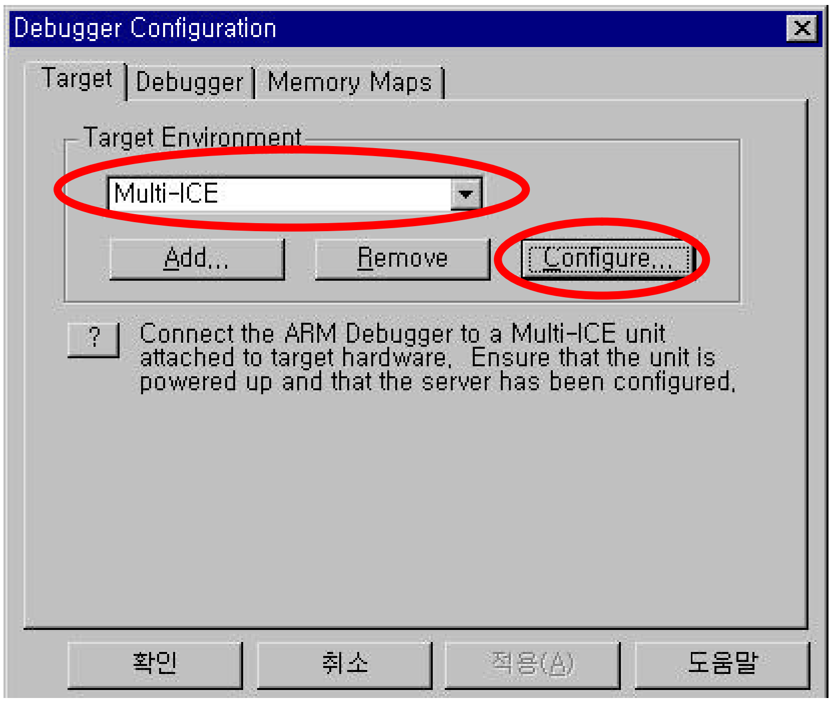
Target Environment

ARMulate

Use the ARM Debugger with the 'ARMulator' Instruction Set Simulator. This allows you to execute ARM programs without physical ARM hardware, by simulating the ARM instructions in software.

<Figure 2-8-1> Debugger Configuration Box

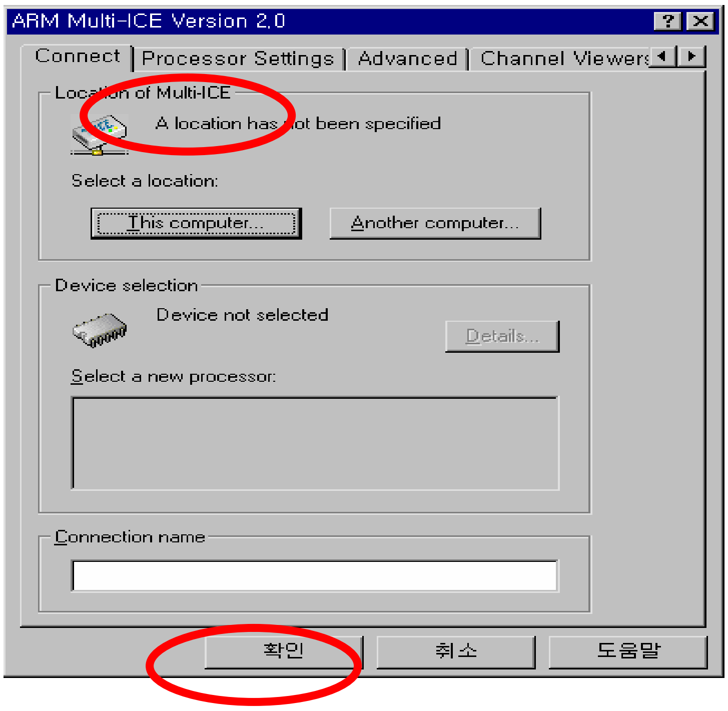
If the "Multi-ICE" item cannot be found in the Target Environment combo list, select "Add" in the Debugger Configuration Box and open Multi-ICE.dll in the Multi-ICE folder.



Connect the ARM Debugger to a Multi-ICE unit attached to target hardware. Ensure that the unit is powered up and that the server has been configured.

<Figure 2-8-2> Debugger Configuration Box

After that, select the "Multi-ICE" item in the Target Environment combo list. Once you click on the "Configure" button, the following dialog box will appear.



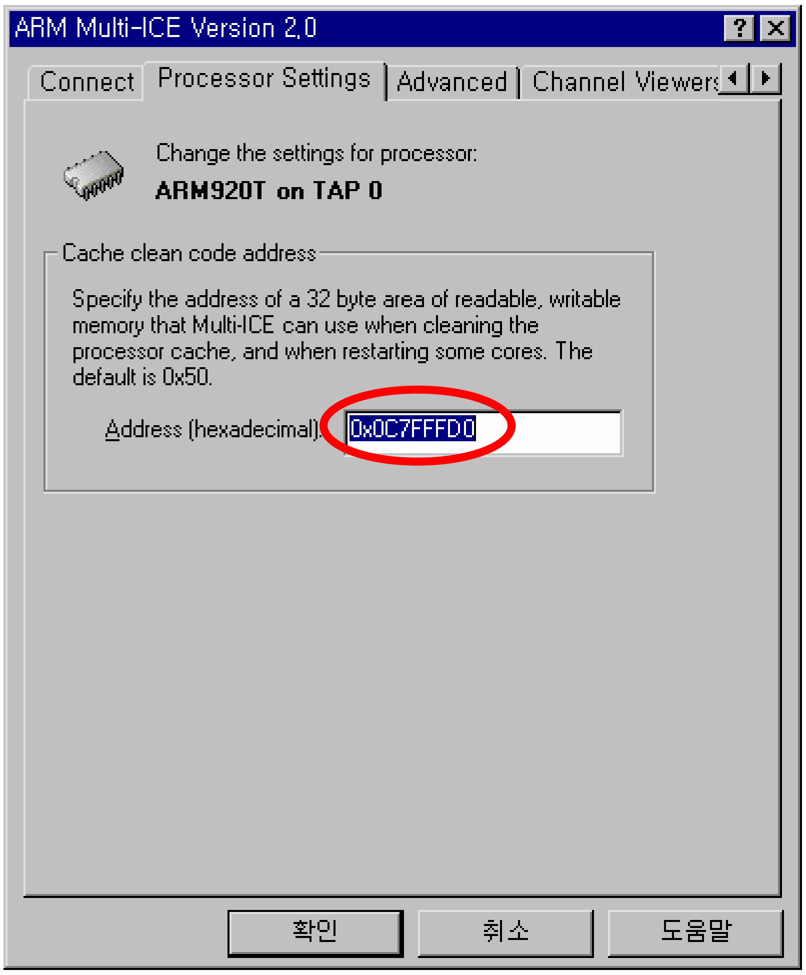
A location has not been specified

Device not selected

<Figure 2-9-1> Multi-ICE Debugger Configuration Box

From the above dialog box, select the Processor Settings tab.

위의 대화창에서 “Processor Settings” 탭을 선택한다.



Change the settings for processor:

ARM920T on TAP 0

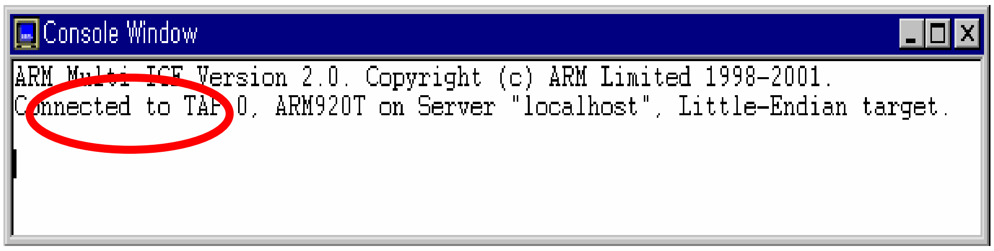
Cache clean code address

Specify the address of a 32 byte area of readable, writable memory that Multi-ICE can use when cleaning the processor cache, and when restarting some cores. The default is 0x50.

Address (hexadecimal): 0x0C7FFFD0

<Figure 2-9-2> Multi-ICE Debugger Configuration Box

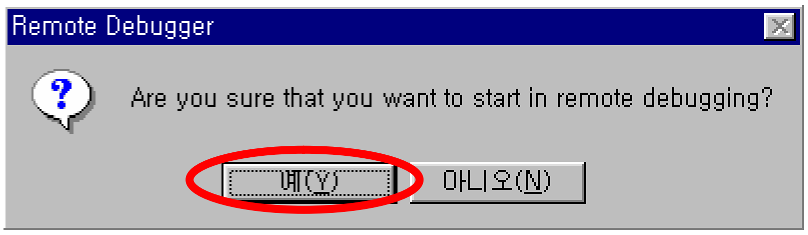
Fill the above address blank with 0x0c7fffd0 and finish the environment configuration by pressing Okay. Then, ADW will be attempting to make a connection with Multi-ICE. If a connection has been successfully made, “Console Window” will appear with the following message,



```
ARM Multi-ICE Version 2.0. Copyright (c) ARM Limited 1998-2001.
Connected to TAP 0, ARM920T on Server "localhost", Little-Endian target.
```

<Figure 2-10> ADW Console Window showing a successful connection with Multi-ICE

In the future you can select "Yes" from the Remote Debugger window when running ADW, unless you want to change the environment configuration.


Remote Debugger

Are you sure that you want to start in remote debugging?

<Figure 2-11> Message Window when Running ADW (Select "Yes")

You can also connect Multi-ICE with serial and parallel cables. Please see Multi-ICE User Manual and ADW User Manual.

Once ADW and Multi-ICE are connected, you can proceed on to the next. Copy the arm9.ini file provided by GamePark to a folder where ADW.EXE exists. ADW is located in the subordinate folder named BIN under the ARM250 folder.

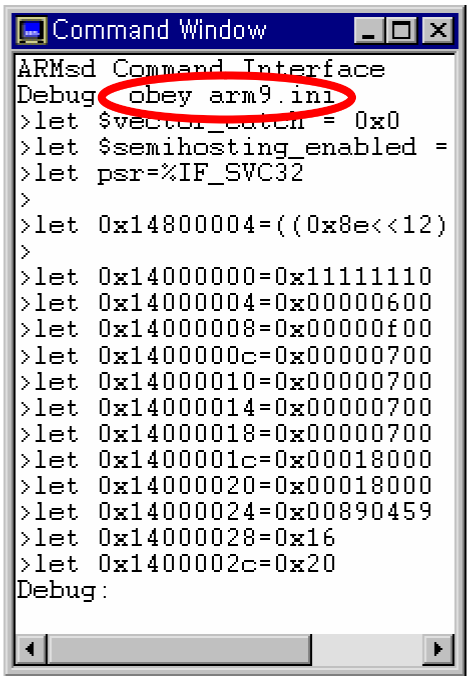
To summerize,

1. Execute ADW(Configure the environment for Multi-ICE.).
2. Go to the options->Configur Debugger->Target Environment->Add to load Multi-ICE.dll in Multi-ICE.
3. Go to the Target Environment->Configure->Processor Settings and enter "0x0c7fffd0" in the address blank.

Next is an example downloading a GP32 program directly onto GP32 through ADW and running it.

3.4 Connecting ADW to GP32

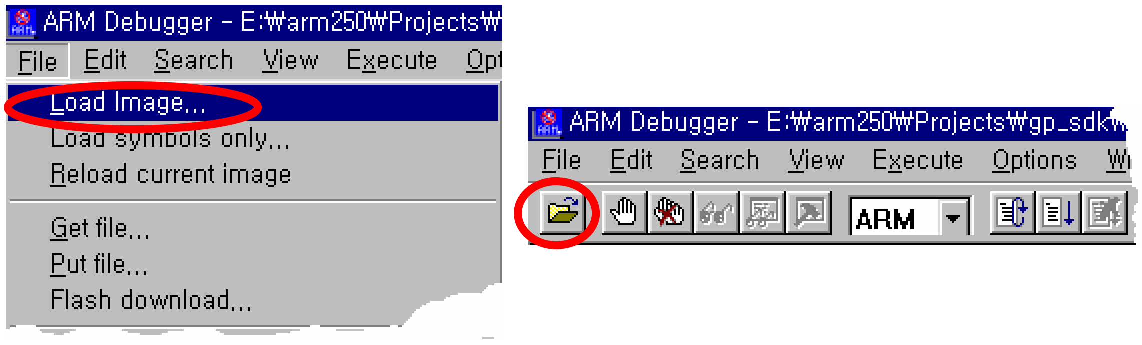
First of all, run ADW. Supply power to Multi-ICE and GP32 and check that they are both operating. You will see a Multi-ICE activation message on the Console Window when you run ADW. If no Console Window appears, select Console menu from ADW View menu. Then command the loading of ARM9.INI mentioned afore in the Command Window. The command is obey gp32.ini.



```
ARMsd Command Interface
Debug: obey arm9.ini
>let $vector_catch = 0x0
>let $semihosting_enabled =
>let psr=%IF_SVC32
>
>let 0x14800004=((0x8e<<12)
>
>let 0x14000000=0x11111110
>let 0x14000004=0x00000600
>let 0x14000008=0x00000f00
>let 0x1400000c=0x00000700
>let 0x14000010=0x00000700
>let 0x14000014=0x00000700
>let 0x14000018=0x00000700
>let 0x1400001c=0x00018000
>let 0x14000020=0x00018000
>let 0x14000024=0x00890459
>let 0x14000028=0x16
>let 0x1400002c=0x20
Debug:
```

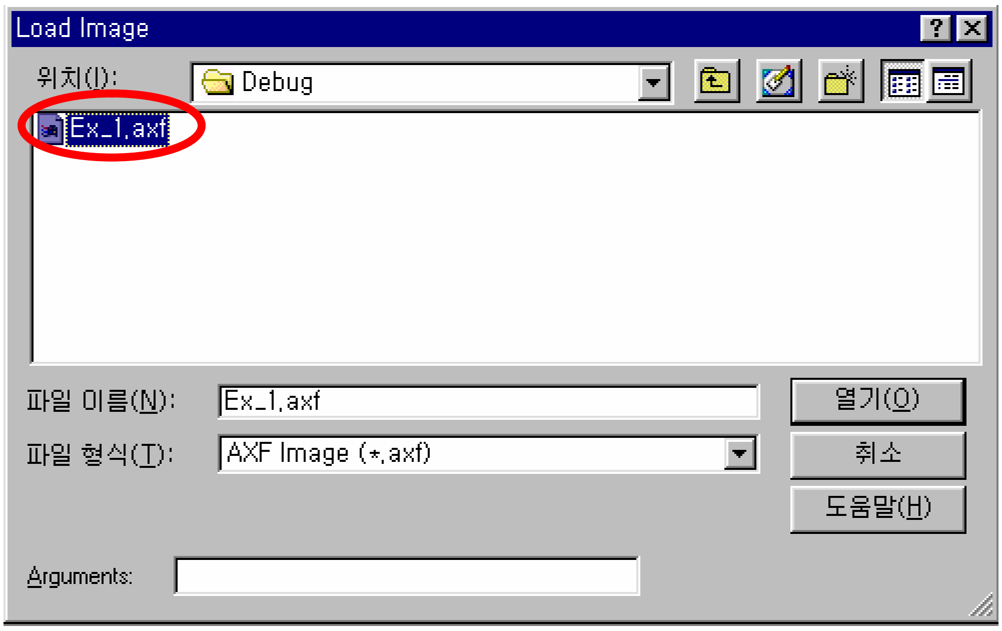
<Figure 2-12> ADW Command Window with Command for GP32 Initialization

Numerous Let commands will be transmitted to GP32 through ADW and Multi-ICE for initialization. If no Command Window appears, select Console menu from ADW View menu. Then load Ex_1.axf created on APM(ARM Project Manager) onto GP32. Command by selecting Load Imag… inside ADW File menu (or by selecting from the toolbar).


ARM Debugger - E:₩arm250₩Projects₩
File Edit Search View Execute Opt
Load Image...
Load symbols only...
Reload current image
Get file...
Put file...
Flash download...
ARM Debugger - E:₩arm250₩Projects₩gp_sdk₩
File Edit Search View Execute Options Wi
ARM

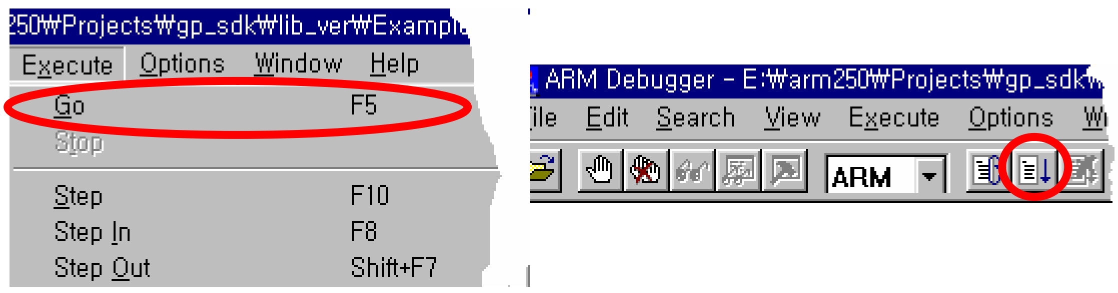
<Figure 2-13> Menu/Tool to Command AXF File Load

A message window to search program image file will appear. Find and select Ex_1.axf.



<Figure 2-14> Message Window to Load AXF File

The program can be executed once the loading has been completed.
Select Execute from the menu, press F5 hot key, or select from the toolbar menu in order to run program.



<Figure 2-15> Menu/Tool to Command Program Execution

You will see a red square on GP32's LCD screen. Up to now, you have been through one development process.

4. Graphic System (I)

4.1 Overview

This chapter explains the GP32 graphic system with examples. As mentioned afore, one of GP32's main features is the high-resolution (320x240) screen. This sets the best environment for game graphic system as the screen corresponds with the internal memory to allot memory areas and show correspondence with the LCD screen. The graphic system supports 256 concurrent colors in 8-bit palette mode and also a maximum of 65,536 colors in 16-bit true color mode.
Graphic System (I) will begin with the basics of LCD graphic system, such as creating surface corresponding with the LCD screen, expressing colors and creating double surface. Graphic System (II) will take you through game-related images. This will be dealt with after looking at GP32 input devices as you can understand more effectively with some knowledge of key control.

4.2 Creating LCD Screen Surface

Drawing squares on GP32 LCD screen will explained with the examples in **Ex_1_Sur** folder. It consists of the 5 following steps.

First, create a surface corresponding to GP32 screen.

This is possible with the **GpLcdSurfaceGet**() function and **GPDRAWSURFACE** structure body variable.

Second, color the newly created surface with the color white.

This is carried out with the **GpRectFill**() function.

Third, set the new surface onto the LCD screen.

This uses the **GpSurfaceSet()** function.

Fourth, enable the LCD.

This is made with the **GpLcdEnable()** function.

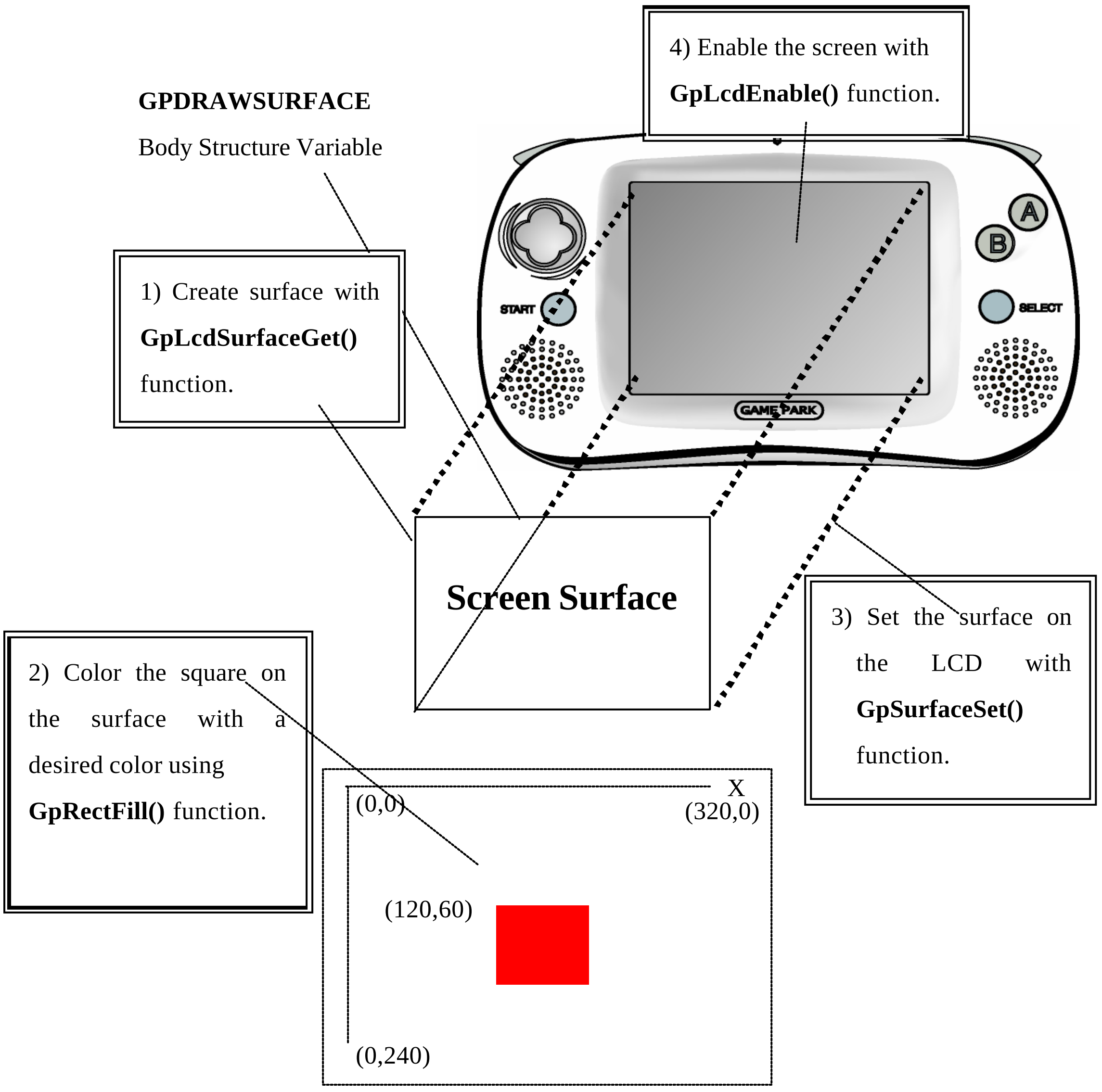
Fifth, draw a square of desired size and color on the surface corresponding to the LCD.

As in the second step, this also uses the **GpRectFill()** function.

```
#include "gpdef.h"
#include "gpstdlib.h"
#include "gpgraphic.h"
#include "gpmain.h"

GPDRAWSURFACE gpDraw;

void GpMain(void * arg)
{
        //GpGraphicModeSet(8, NULL);
        GpLcdSurfaceGet(&gpDraw, 0);
       //surface fill color in white
        GpRect Fill(NULL, &gpDraw, 0, 0, gpDraw.buf_w, gpDraw.buf_h, 0xff);
        //make gpDraw with primary surface
        GpSurfaceSet(&gpDraw);
       //Draw a 60x60 red box on the generated surface.
        GpRectFill(NULL, &gpDraw, 120, 60, 80, 80,0xe0);
        while(1)
          ;
}
```

<List 3-1> Example Source of Creating Screen Surface

The following is a diagram explaining the concept of the list above.


4) Enable the screen with **GpLcdEnable()** function.
GPDRAWSURFACE
Body Structure Variable
1) Create surface with **GpLcdSurfaceGet()** function.
START
SELECT
A
B
GAME PARK
Screen Surface
3) Set the surface on the LCD with **GpSurfaceSet()** function.
2) Color the square on the surface with a desired color using **GpRectFill()** function.
X
(0,0)
(320,0)
(120,60)
(0,240)

<Figure 3-1> Diagram of LCD Screen and Surface Creation

The logical coordinates of such screen surface are as follows. Top-left is the origin of the coordinates, with the X axis running on the right and the Y axis going downwards.
Therefore, the above codes will result in a red square of (W,H) = (80,80) on a white screen with coordinates (X,Y) = (120,60).
This process will be explained in more details later on.

Note:

GP32 internally generates surface buffers that can become the primary buffer of the screen and manages them. A developer has to get the surface buffers previously generated by the firmware. If you directly allocate memory to generate a buffer, the buffer cannot become a primary buffer. The firmware generates four surface buffers in 8-bit mode and two surface buffers in 16-bit mode.

In order to create a surface, a variable of **GPDRAWSURFACE** structure body is necessary. Create variable gpDraw with such functions on the structure body variable declaration in line 6 of the above list.

```
6:      GPDRAWSURFACE gpDraw;
```

The definition of GPDRAWSURFACE structure body is in the gpgraphic.h file inside the GPINCLUDE folder. Let's take a look.

```
typedef struct tagGPDRAWSURFACE{
        unsigned char * ptbuffer;     // buffer start address
        int bpp;                      // reserved
        int buf_w;                    // buffer width
        int buf_h;                    // buffer height
        int ox;                       // buffer origin x co-ordinate
        int oy;                       // buffer origin y co-ordinate
        unsigned char * o_buffer;
}GPDRAWSURFACE;
```

The actual screen surface is returned by the GpLcdSurfaceGet function and this surface information is put into the GPDRAWSURFACE structure.

.

```
 int GpLcdSurfaceGet(GPDRAWSURFACE * ptgpds, int idx);

/*
        <arg 1>     GPDRAWSURFACE * ptgpds
                    The address of structure including the information about the surface to be generated
        <arg 2>     int idx
                    The index value of the surface to be generated
        <return>    int
                    If the surface generation is successful, this function will return GPC_DRAW_Ok, if not, it returns
                    GPC_DRAW_ERR.
*/
```

When the surface is created, color it white with GpRectFill() function.

```
14:         GpRectFill(NULL, &gpDraw, 0, 0, gpDraw.buf_w, gpDraw.buf_h, GPC_SC_WHITE);
```

Line 14 of the example list goes through this process. It is necessary before setting the surface to the LCD screen in order to place your desired picture on it.
Below is a detailed explanation about the GpRectFill(…) function.

```
int GpRectFill(GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,
                        int dx,int dy,int width,int height,
                    unsigned char color);
/*
    <arg 1>    GPDRAWTAG * gptag
               Address of the structure body variable representing the clipping area.
    <arg 2>    GPDRAWSURFACE * ptgpds
               Address of the structure body variable containing data on the surface where the square is to be drawn.
    <arg 3>    int dx
               x coordinate for the starting point of square.
    <arg 4>    int dy
               y coordinate for the starting point of square.
    <arg 5>    int width
               Width of square.
    <arg 6>    int height
               Height of square.
    <arg 7>    unsigned char color
               Color of square.
    <return>   int
               No meaning.
*/
```

Once, the new surface is initialized with the color white, the gpDraw surface is set on the LCD with GpSurfaceSet() function. This function actually connects the surface to the LCD.

```
16:     GpSurfaceSet(&gpDraw);     //make gpDraw with primary surface
```

Line 16 proceeds as below.

```
void GpSurfaceSet(GPDRAWSURFACE * ptgpds);
/*
    <arg 1>    GPDRAWSURFACE * ptgpds
               Address of structure body variable containing data on the surface to be set on the LCD.
*/
```

The LCD screen will now be colored white.

The next step uses GpRectFill() to color the square in red.

```
18:     GpRectFill(NULL, &gpDraw, 120, 60, 80, 80, 0xe0);
```

Line 18 draws the red square.

The next code is the waiting code.

The structure body and functions used in the example above will be used over and over again, so you must learn them carefully.

In other words, you need to learn the structure body GPDRAWSURFACE and functions GpLcdSurfaceGet(), GpRectFill(), GpSurfaceSet().

4.3 8-Bit Palette Mode / 16-Bit True Color Mode

The GP32 graphic system supports 16-bit colors and its graphics design is almost the same as the one for the existing PC game (8-bit, 16-bit).

The GP32 color format has been illustrated as follows:

RED[5bit] : GREEM[5bit] : BLUE[5bit] : Intensity[1bit

※ Precaution

While the color LUT architecture of PC's VGA graphic card has the 24-bit color resolution, GP32 has the 16-bit resolution. Therefore, each palette entry of GP32 also has the 16-bit resolution in 8-bit mode meaning it has limited colors. For instance, RGB(0,0,0) is separately distinguishable from RGB(7,7,7) on PC, but not on GP32. In other words, GP32 mistakes RGB(7,7,7) for RGB(0,0,0). Consequently, A developer should single out each 32 colors among **256** levels of **red**, green and blue.

4.4 Creating Double Surface

Double surface technique consists of creating two surfaces corresponding to the screen. One is used for the present screen and the other for the new screen. The two are shown alternately to bring a change on the screen. This technique is often used to make the screen smooth for games. This process on GP32 is shown below.

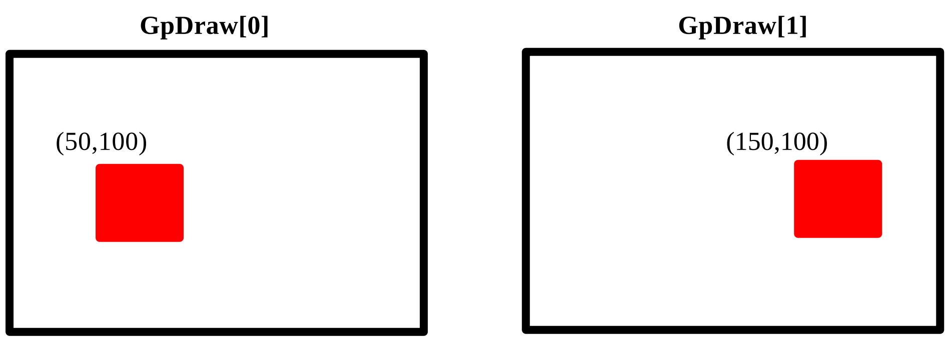
```
#include "gpdef.h"
#include "gpstdlib.h"
#include "gpgraphic.h"
#include "gpmain.h"

GPDRAWSURFACE gpDraw[2];
int nflip;

void GpMain(void * arg)
{
        int i;
        unsigned int n_tick;

        i = GpLcdSurfaceGet(&gpDraw[ 0 ],0);
        i = GpLcdSurfaceGet(&gpDraw[ 1 ],1);
        GpRectFill(NULL, &gpDraw[0], 0, 0, gpDraw[ 0].buf_w, gpDraw[0].buf_h, 0xff);
        GpRectFill(NULL, &gpDraw[1], 0, 0, gpDraw[ 1].buf_w, gpDraw[1].buf_h, 0xff);
        GpSurfaceSet(&gpDraw[0]);  //sets gpDraw[0] to the primary surface
        nflip = 1;    //gpDraw[1]이 This function indicates that gpDraw[0] is the back surface.

        while(1)
        {
                /*Draw 100x100 red box at the coordinates (nflip*100, 0) of the back surface.
                GpRectFill(NULL, &gpDraw[nflip], nflip * 100, 100, 100, 100, 0xe0);
                GpSurfaceFlip(&gpDraw[nflip++]);
        nflip %= 2;   //Save the reference index of the previous primary surface as nflip, the reference value of the back surface.
        /*About 1000msec of time delay*/
        n_tick = GpTickCountGet();

        while ( ( GpTickCountGet() - n_tick ) < 1000 )
            ;
    }
}
```

<List 3-3> Example Source of Creating Double Surface

The above list contains codes for creating two screen surfaces, drawing a red square on both surfaces, and showing them alternately for 1 second.

The next codes are GPDRAWSURFACE declaring these 2 surfaces and their respective index variable declaration.

```
6:    GPDRAWSURFACE gpDraw[2];
7:    int nflip;
```



<Figure 3-3> Screens Corresponding to Double Surface

The codes now show how the surfaces are actually created and colored white.

```
14:        i = GpLcdSurfaceGet(&gpDraw[0],0);
15:        i = GpLcdSurfaceGet(&gpDraw[1],1);
16:        GpRectFill(NULL, &gpDraw[0], 0, 0, gpDraw[0].buf_w, gpDraw[0].buf_h, 0xff);
17:        GpRectFill(NULL, &gpDraw[1], 0, 0, gpDraw[1].buf_w, gpDraw[1].buf_h, 0xff);
18:        GpSurfaceSet(&gpDraw[0]);  //Set gpDraw[0] as gpDraw[0]을 primary surface 로 setting
```

These codes are for setting gpDraw[0] on the LCD and enabling the LCD.

```
18:        GpSurfaceSet(&gpDraw[0]);  //set gpDraw[0] as the primary surface
                                      gpDraw[0]을 primary surface 로 setting
```

The LCD will now be white.

```
19:        nflip = 1;    //indicate that gpDraw[1] is the back surface
20:
21:        while(1)
22:        {
23:            /*draw a red square of 100*100 at the coordinate (nflip*100, 0) of the back surface*/
24:            GpRectFill(NULL, &gpDraw[nflip], nflip * 100+50, 100, 100, 100, GPC_SC_RED);
25:            GpSurfaceFlip(&gpDraw[nflip++]); // Change back surface to primary surface
26:            nflip %= 2;   //save reference value of previous primary surface as nflip, the reference index of
                             back surface
27:            /* time delay for about 1000msec */
28:            n_tick = GpTickCountGet();
29:
30:            while ( ( GpTickCountGet() - n_tick ) < 1000)
31:                ;
32:        }
```

The above codes will draw the squares on each surface in different places and show them alternately for 1 second. Thus, two different squares will be shown on the LCD.

The **GpSurfaceFlip()** function alternates the LCD screen which is carried out in Line 25 of the example.

```
        GpSurfaceFlip(&gpDraw[nflip++]); // change back surface to primary surface
```

Variable nflip is the index that alternates the 2 surfaces by repeating 0 and 1. A detailed explanation of **GpSurfaceFlip(…)** is as follows.

```
void GpSurfaceFlip(GPDRAWSURFACE * ptgpds);
/*      <arg 1>     GPDRAWSURFACE * ptgpds
                    Address of the structure body of the surface to be displayed.
*/
```

The next codes makes each surface last 1 second.

```
        n_tick = GpTickCountGet();

        while ( ( GpTickCountGet() - n_tick ) < 1000 )
```

GpTickCountGet() is the function showing the value of current system tick counter. 1 Tick represents 1 msec, so 3000 msec will be 3 seconds.

Below is the definition of **GpTickCountGet(…)**.

```
unsigned int GpTickCountGet(void);
/*      <return>    Range of the system's present tick counter is 0x00000000~0xFFFFFFFF.
*/
```

3.5 Character Output

Function for character output is **GpTextOut()**. Use it to output characters of desired color wherever you want.

```
#include "gpdef.h"
#include "gpstdlib.h"
#include "gpgraphic.h"
#include "gpmain.h"

GPDRAWSURFACE gpDraw;

void GpMain(void * arg)
{
        GpLcdSurfaceGet(&gpDraw, 0);

        GpRectFill(NULL, &gpDraw, 0, 0, gpDraw.buf_w, gpDraw.buf_h, 0xff);

        GpSurfaceSet(&gpDraw);


        GpTextOut(NULL, &gpDraw, 10, 10, (char*)"GP32 TextOut Example!", 0x00);
        GpTextOut(NULL, &gpDraw, 10, 40, (char*)" 1.Example1\n 2.Example2\n 3.Example3",
                        0xe0);

        while(1)
                ;
}
```

<List 3-4> Example Source of Character Output

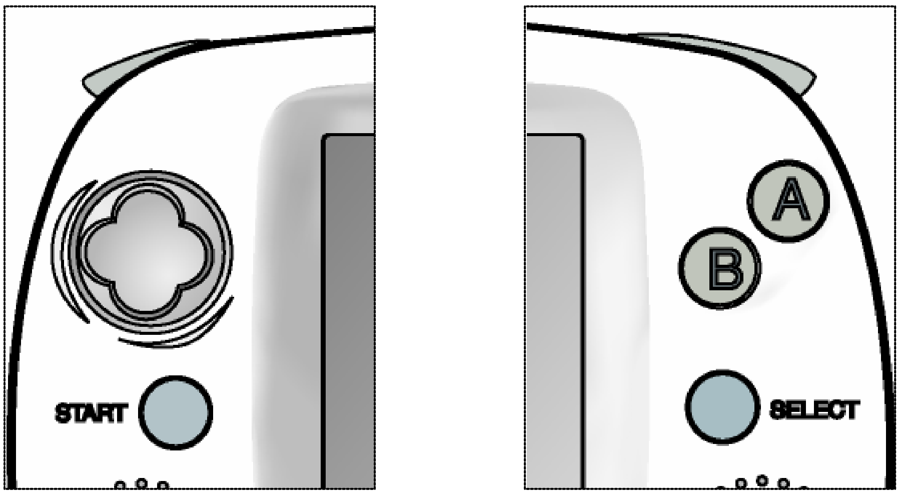
The above example is very simple. It will be easier to understand once the user learns how to use GpTextOut() it. Below is the definition of GpTextOut(…).

```
void GpTextOut(GPDRAWTAG * gptag, GPDRAWSURFACE * ptgpds, int x, int y,
                    unsigned char* sout, unsigned char color);
/*
        <arg 1>     GPDRAWTAG * gptag
                    Address of structure body variable designating the clipping area of output
        <arg 2>     GPDRAWSURFACE * ptgpds
                    Address of structure body variable designating the surface for output
        <arg 3> int x
                    x coordinate for starting output
        <arg 4> int y
                    y coordinate for starting output
        <arg 5> unsigned char * sout
                    Address of variable containing string for output
        <arg 6> unsinged char color
                    Value designating the color of string
*/
```

5. Input System

5-1. Overview

The GP32 input system is comprised of a joystick on the left and function buttons on the right.



<Figure 4-1> GP32 Input System

The START button is located on the left of the GP32 as a function key and in the top left there is the LEFT button. The GP32 has precise directional control with the 8-way directional joystick (top, bottom, left, right, left top, left bottom, right top, right bottom). In addition, the GP32 has 3 function keys (A, B, Selection) and the RIGHT button on the right.

The LEFT key on the top left is recognized as F2 (presently FL), the RIGHT key on the top right as F1 (presently FR), the A key on the right as ENTER (presently FA), the B key as F3 (presently FB) and the START and SELECT buttons as respectively START and SELECT in your programming languages.

5.2 Joystick and Function Button Input

Let’s take a look into examples of programming the joystick and function buttons which are the most important input devices.

```
#include "gpdef.h"
#include "gpstdlib.h"
#include "gpgraphic.h"
#include "gpmain.h"
#include "gpfont.h"
#include "gpmain.h"

GPDRAWSURFACE gpDraw[2];
int nflip;

void GpMain (void *arg)
{
        int i;
        unsigned int n_tick;
        unsigned char keydata;
        int pos_x, pos_y;
        int ExKey;
        for ( i = 0 ; i < 2 ; i++)
        {
                GpLcdSurfaceGet(&gpDraw[i], i);
                GpRectFill(NULL, &gpDraw[i], 0, 0, gpDraw[i].buf_w, gp Draw[i].buf_h, 0xff);
        }

        GpSurfaceSet(&gpDraw[0]);   //This function sets the gpDraw[0] to the primary surface.

        nflip = 1;    //gpDraw[1]이 This function indicates gpDraw[1] is the back surface.

        while(1)
        {
                pos_x = 0;
                pos_y = 0;

                 GpKeyGetEx(&ExKey);
                keydata = ExKey & 0xff;

                if ( ( keydata & GPC_VK_LEFT ) == GPC_VK_LEFT )
                {
                        GpTextOut(NULL, &gpDraw[nflip], pos_x, pos_y, (char*)"LEFT key pressed", 0x0);
                        pos_y += 20;
                }
                if ( ( keydata& GPC_VK_UP ) == GPC_VK_UP )
                {
                        GpTextOut(NULL, &gpDraw[nflip], pos_x, pos_y, (char*)"UP key pressed", 0x0);
                        pos_y += 20;
                }
                if ( ( keydata & GPC_VK_RIGHT ) == GPC_VK_RIGHT )
                {
                        GpTextOut(NULL, &gpDraw[nflip], pos_x, pos_y, (char*)"RIGHT key pressed", 0x0);
                        pos_y += 20;
                }
                if ( ( keydata & GPC_VK_DOWN ) == GPC_VK_DOWN )
                {
                        GpTextOut(NULL, &gpDraw[nflip], pos_x, pos_y, (char*)"DOWN key pressed", 0x0);
                        pos_y += 20;
                }
                if ( ( keydata & GPC_VK_F1 ) == GPC_VK_F1 )
                {
                        GpTextOut(NULL, &gpDraw[nflip], pos_x, pos_y, (char*)"UP RIGHT key pressed", 0x0);
                        pos_y += 20;
                }
61:                 if( ( keydata & GPC_VK_F2 ) == GPC_VK_F2 )
62   :              {
63:                       GpTextOut(NULL, &gpDraw[nflip], pos_x, pos_y, (char *)"UP LEFT key pressed", 0x0);
64:                       pos_y += 20;
65:                 }
66:                 if ( ( keydata & GPC_VK_F3) == GPC_VK_F3 )
67:                 {
68:                       GpTextOut(NULL, &gpDraw[nflip], pos_x, pos_y, (char *)"B key pressed" , 0x0);
69:                       pos_y += 20;
70:                 }
71:                 if ( ( keydata & GPC_VK_ENTER) == GPC_VK_ENTER )
72:                 {
73:                       GpTextOut(NULL, &gpDraw[nflip], pos_x, pos_y, (char *)"A key pressed" , 0x0);
74:                       pos_y += 20;
75:                 }
76:                 if (ExKey & GPC_VK_START)
77:                 {
78:                       GpTextOut(NULL, &gpDraw[nflip], pos_x, pos_y, (char *)"START key pressed" , 0x0);
79:                       pos_y += 20;
80:                 }
81:                 if (ExKey & GPC_VK_SELECT)
82:                  {
83:                       GpTextOut(NULL, &gpDraw[nflip], pos_x, pos_y, (char *)"SELECT key pressed" , 0x0);
84:                       pos_y += 20;
85:                 }
86:
87:                 GpSurfaceFlip(&gpDraw[nflip++]);
88:                 nflip %= 2;
89:
90:                 /*about 400msec of time delay*/
91:                 n_tick = GpTickCountGet();
92:                 while ( ( GpTickCountGet() - n_tick ) < 400)
93:                 ;
94:           }
95:   }
```

<List 4-1-1> Example source to output a message

to the screen based on the input key value

The key API in the examples listed above is the GpKeyGetEx(). This function returns the current state of the GP32 joystick and buttons, which displays the current input status on the LCD. It defines the variable to get the input key value in the 15th and 17th lines of the list and in the 33rd and 34th lines it actually gets the input value.

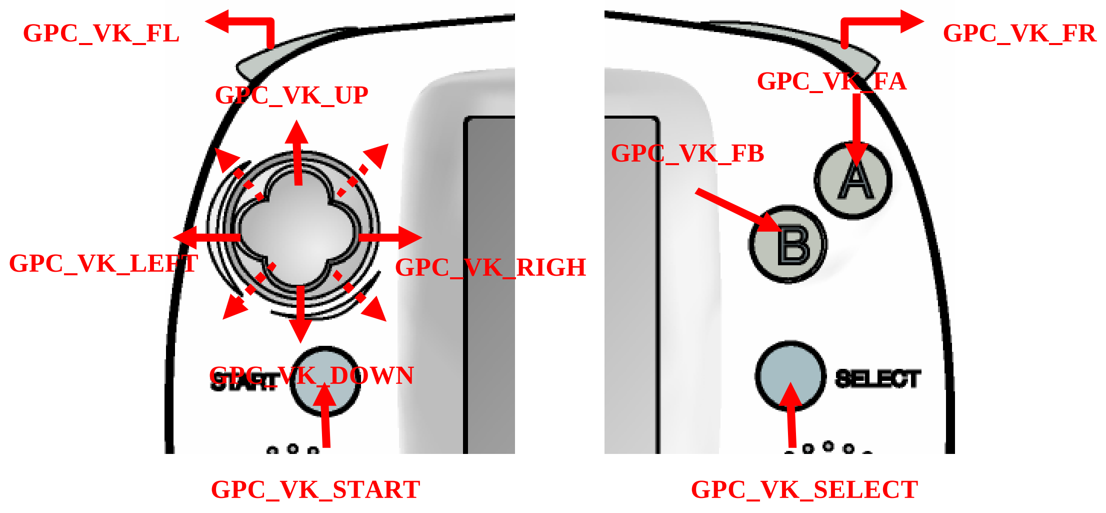
```
15:         unsigned char keydata;
17:         int ExKey;

33:         GpKeyGetEx(&ExKey);
34:         keydata = ExKey & 0xff;
```

Finally, the "if clauses" in the 35th through 85th lines output the text based on these key values. Each value for the state property is defined in the gpdef.h file.

```
#define     GPC_VK_NONE       0x00
#define     GPC_VK_LEFT       0x01
#define     GPC_VK_UP         0x08
#define     GPC_VK_RIGHT      0x04
#define     GPC_VK_DOWN       0x02
```

```
#define    GPC_VK_START      0x100
#define    GPC_VK_SELECT     0x200

#define    GPC_VK_FA         GPC_VK_ENTER     #define    GPC_VK_ENTER     0x40 //AT OLD, VK_F1
#define    GPC_VK_FB         GPC_VK_F3        #define    GPC_VK_F3        0x20 //AT OLD, VK_F2
#define    GPC_VK_FL         GPC_VK_F2        #define    GPC_VK_F2        0x10 //AT OLD, VK_F3
#define    GPC_VK_FR         GPC_VK_F1        #define    GPC_VK_F1        0x80 //AT OLD, VK_ENTER
```

The constant value corresponding to the input key has been defined in the following figure.



<Figure 4-2> Defined constant values corresponding to input keys

```

        while(1)
        {
            keydata = GpKeyGet();

            if ( GpKeyChanged() & 0xff)
            {
                keydata = GPC_VK_NONE;
            }
            if ( keydata& GPC_VK_F3 )
            {
                GpTextOut(NULL, &gpDraw[nflip], pos_x, pos_y, (char*)"B key pressed", 0x0);
                pos_y += 20;
            }
            else if(keydata& GPC_VK_F1 )
            {
                GpTextOut(NULL, &gpDraw[nflip], pos_x, pos_y, (char*)"UP key pressed", 0x0);
                pos_y += 20;
            }
```

<List 4-1-2> An example of using the GpKeyChanged() function

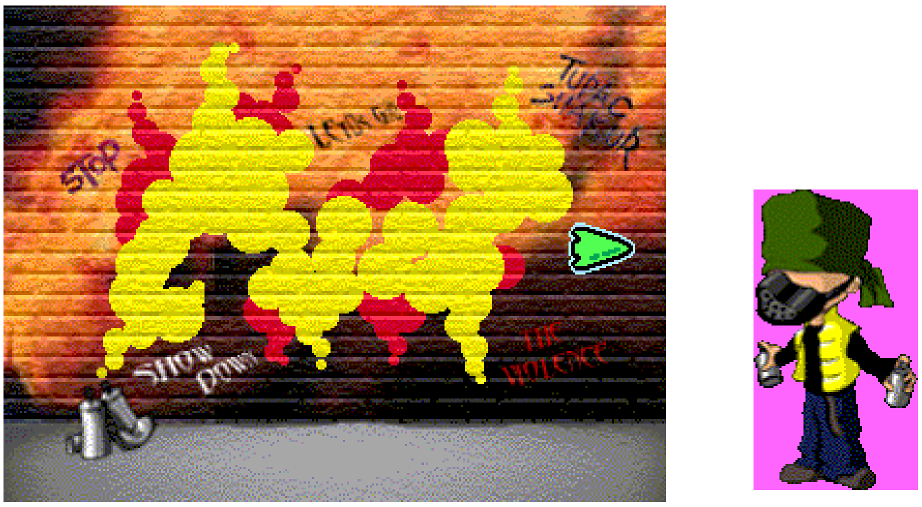
There is a close correspondence between the GpKeyChanged() function and the GpKeyGet() function. The GpKeyChanged() function is the variable to indicate the changes made to the input key status between each GpKeyGet() function calls.

After comparing the KEY values returned when the GpKeyGet and GpKeyGetEx functions were lately called with the latest KEY values, this function performs a bit-set operation on the relevant

KEY value changes and returns the result.

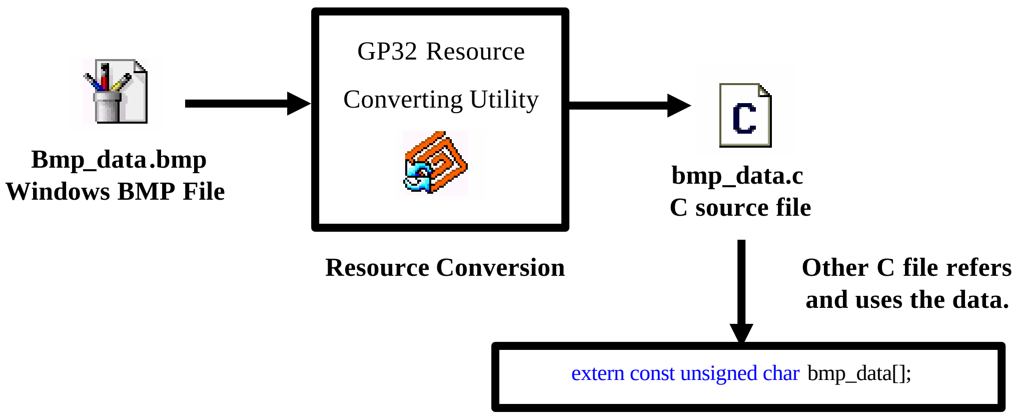
6. Graphic System (II)

This chapter explains based on simple examples starting with how to output image files which are useful in actual game programs onto GP32's LCD, then how to output character sprites, and finally how to move the output positions of images using key input.
The 2 image files below will be used as examples.
One is an image of 320x240 for the background and the other is character sprite of 82x150.



<Figure 5-1> Background and Character Sprite (L: Background, R: Character)

The image files above are in Windows BMP file, thus you would need to convert them into certain files using GP32 Resource Converting Utility in order to use such image resources in the program. Here, they have been converted to C source file.
(See GP32 Dev Utility for more detail.)



```
extern const unsigned char bmp_data[];
```

<Figure 5-2> Process of Converting Actual Image for the Program

6.1 Drawing Sprite 1

This is an example of putting a sprite of 82x150 on a background of 320x240 on the LCD.

```
#include "gpdef.h"
#include "gpstdlib.h"
#include "gpgraphic.h"
#include "gpmain.h"

GPDRAWSURFACE gpDraw[2];
int nflip;

extern const unsigned int img_back[320][60];    //size (320*240)
extern const unsigned int img_char[80][38];     //size (80*150)

void GpMain(void* arg)
{
        int i;
        unsigned int n_tick;
        unsigned char keydata;
        int pos_x, pos_y;
        for ( i = 0 ; i < 2 ; i++)
        {
                GpLcdSurfaceGet(&gpDraw[i],i);

                GpRectFill(NULL, &gpDraw[i], 0, 0, gpDraw[i].buf_w, gpDraw[i].buf_h,0xff);
        }
        GpSurfaceSet(&gpDraw[0]);  //This function sets the gpDraw[0] to the primary surface.

        nflip = 1;   //This function indicates that gpDraw is the back surface.
        pos_x = 120;
        pos_y = 80;
        while(1)
        {
                //copy a background image to the back surface
                GpBitBlt(NULL, &gpDraw[nflip], 0, 0, 320, 240, (unsigned char*)img_back, 0, 0, 320, 240);
                //Send a character image with no transparent color to the (0,0) coordinates of the background image.
                GpBitBlt(NULL, &gpDraw[nflip], 0, 0, 80, 150, (unsigned char*)img_char, 0, 0, 80, 150);
                keydata = GpKeyGet();
                // The position value of the character varies by the KEY input value.
                if ( keydata & GPC_VK_LEFT )
                        pos_x--;
                if ( keydata & GPC_VK_UP )
                        pos_y--;
                if ( keydata & GPC_VK_RIGHT )
                        pos_x++;
                if ( keydata & GPC_VK_DOWN )
                        pos_y++;
                //Send a character sprite which has a transparent color (0xef) on the background image of the back
                surface.
                GpTransBlt(NULL, &gpDraw[nflip], pos_x, pos_y, 80, 150,
                                (unsigned char*)img_char, 0, 0, 80, 150, 0xef);
                GpSurfaceFlip(&gpDraw[nflip++]);
                nflip %= 2;
                /*about 10msec of time delay*/
                n_tick = GpTickCountGet();
                while ( ( GpTickCountGet() - n_tick ) < 10)
                        ;
        }
}
```

<List 5-1> Example Source of Sprite Output (I)

In addition, get joystick values and make the sprite move up, down, right, and left. A double buffer is used to make the movements natural.

Firstly, define the data of 320x240 background and 82x150sprite in img_background.c and img_sprite.c files respectively. How to prepare such image data will be explained later. Declare in gpmain.c as shown below.

```
extern const unsigned int img_background[320][60];    //size (320*240)
extern const unsigned int img_sprite[82][38];      //size (82*150)
```

Complete the initial wok of creating double surface. The draw the background inside the while loop and the sprite, including the transparent color, on the back surface.

```
        //copy a background image to the back surface
        GpBitBlt(NULL, &gpDraw[nflip], 0, 0, 320, 240, (unsigned char*)img_back, 0, 0, 320, 240);
        //Send a character image with no transparent color to the (0,0) coordinates of the background image.
        GpBitBlt(NULL, &gpDraw[nflip], 0, 0, 80, 150, (unsigned char*)img_char, 0, 0, 80, 150);
```

This can be realized with the GpBitBlt() function.

Now, get key input and adjust the position of the sprite.

```
        keydata = GpKeyGet();
        // The position value of the character varies by the KEY input value.
        if ( keydata & GPC_VK_LEFT )
            pos_x--;
        if ( keydata & GPC_VK_UP )
            pos_y--;
        if ( keydata & GPC_VK_RIGHT )
            pos_x++;
        if ( keydata & GPC_VK_DOWN )
            pos_y++;
```

Then, draw the sprite with transparent color on the back surface using GpTransBlt() function.

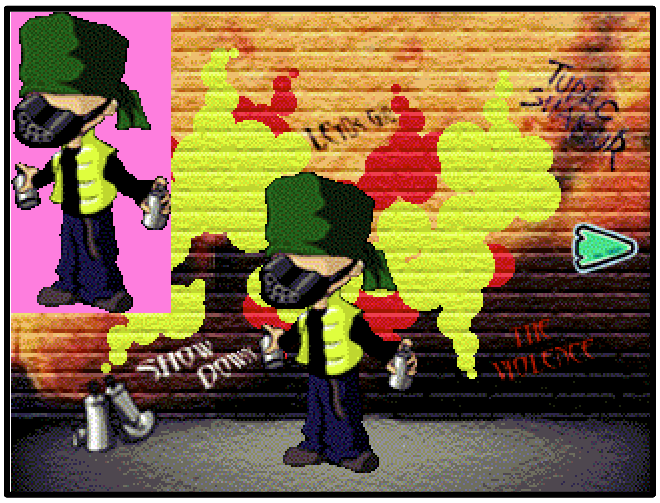
```
        //Send a character sprite which has a transparent color (0xef) on the background image of the back surface.
        GpTransBlt(NULL, &gpDraw[nflip], pos_x, pos_y, 80, 150,
                    (unsigned char*)img_char, 0, 0, 80, 150, 0xef);
```

The following transfers the back surface with the sprite onto the LCD screen.

```
        GpSurfaceFlip(&gpDraw[nflip++]);
        nflip %= 2;
```

The only difference between GpBitBlt() and GpTransBlt() functions is whether the concept of transparent color exists or not, the rest are the same. In other words, GpTransBlt() contains a argument designating the transparent color and its function is not copying the desired color of the image drawn on the surface.

If you follow through the examples above, you will get the screen result shown below. The sprite moves in the direction the joystick moves.



<Figure 5-3> Sprite Output (I)

This is an explanation of GpBitBlt(…) function.

```
/* ?  copy image from buffer to buffer. */

int GpBitBlt(GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,int dx,int dy,
             int width,int height,unsigned char * src, int sx,int sy,int imgw, int imgh);
/*
               GPDRAWTAG * gptag : clipping square (clipping area is 320 * 240 if the value is NULL)
               GPDRAWSURFACE * ptgpds : address of target surface to copy image
               int dx,int dy : starting coordinates (x,y) of target to copy image
               int width,int height : width and height of area to copy image
               unsigned char * src : address of original image buffer
               int sx,int sy : starting coordinates (x,y) to copy original image
               int imgw,int imgh : size of original image
*/
```

This is an explanation of GpTransBlt(..) function.

```
/* ?  copy image without transparent color from buffer to buffer.. */

int GpTransBlt(GPDRAWTAG * gptag,GPDRAWSURFACE * ptgpds,int dx,int dy,
                           int width,int height,unsigned char *src,int sx,int sy,
                           int imgw,int imgh,unsigned char color);
/*
            GPDRAWTAG * gptag : clipping square (clipping area is 320 * 240 if the value is NULL)
            GPDRAWSURFACE * ptgpds : target buffer to copy image
            int dx,int dy : coordinates of target to copy image
            int width,int height : area to copy image
            unsigned char * src : address of original image buffer
            int sx,int sy : Starting coordinates to copy original image
            int imgw,int imgh : size of original image
            unsigned char color: transparent color
*/
```

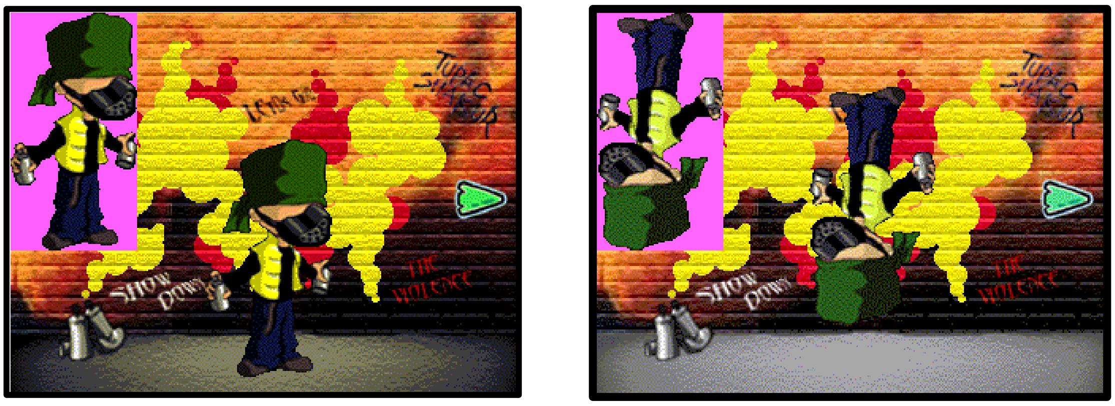
6.2 Drawing Sprite 2

The next we will look at is the examples of using left and right, top and bottom inversion of a sprite and an image.

```
#include "gpdef.h"
#include "gpstdlib.h"
#include "gpgraphic.h"
#include "gpmain.h"

GPDRAWSURFACE gpDraw[2];
int nflip;
extern const  unsigned int img_back[320][60];      //size (320*240)
extern const  unsigned int img_char[80][38];       //size (80*150)

void GpMain(void)
{
        int i;
        unsigned int n_tick;
        unsigned char keydata;
        int pos_x, pos_y;
        int sprite_dir;

        for ( i = 0 ; i < 2 ; i++)
        {
                GpLcdSurfaceGet(&gpDraw[i],i);

                GpRectFill(NULL, &gpDraw[i], 0, 0, gpDraw[i].buf_w, gpDraw[i].buf_h,GPC_SC_WHITE);
        }

        GpSurfaceSet(&gpDraw[0]);  /This function sets the gpDraw[0] to primary surface

        nflip = 1;    //This function indicates that the gpDraw[1] is the back surface.
        pos_x = 120;
        pos_y = 80;
        sprite_dir = 0;

        while(1)
        {
                keydata = GpKeyGet();
                //copy the background image to the back surface.
                GpBitBlt(NULL, &gpDraw[nflip], 0, 0, 320, 240, (unsigned char*)img_back, 0, 0, 320, 240);
                // The position value of a character varies by the KEY input value.
                if ( keydata & GPC_VK_LEFT )
                {
                        sprite_dir = 0;      // The original sprite
                        pos_x--;
                }
                if ( keydata & GPC_VK_UP )
                {
                        sprite_dir = 0;      // The original sprite
                        pos_y--;
                }
                if ( keydata & GPC_VK_RIGHT )
                {
                        sprite_dir =1; //This function performs the left and right mirroring of a spite.
                        pos_x++;
                }
                if ( keydata & GPC_VK_DOWN )
                {
                        sprite_dir =2; // This function performs the top and bottom mirroring of a sprite.
                        pos_y++;
                }
                if(sprite_dir == 0)
                {

                        GpBitBlt(NULL,&gpDraw[nflip], 0, 0, 80, 150, (unsigned char*)img_char, 0, 0, 80, 150);

                        GpTransBlt(NULL, &gpDraw[nflip], pos_x, pos_y, 80, 150,
                                (unsigned char*)img_char, 0, 0, 80, 150, 0xef);
                }
        else if(sprite_dir == 1)
        {
                //Perform the left and right inversion of the character with no transparent color and send it to the
                   (0,0) coordinates of the background image.
                GpBitLRBlt(NULL,&gpDraw[nflip],  0, 0, 80, 150, (unsigned char*)img_char, 0, 0, 80, 150);
                //Perform the left and right inversion of the character sprite and output it on the background image
                   of the back surface
              GpTransLRBlt (NULL, &gpDraw[nflip], pos_x, pos_y, 80, 150, (unsigned char*)img_char,
                                    0, 0, 80, 150, 0xef);
            }
            else if(sprite_dir == 2)
            {
               // Perform the top and bottom inversion of the character with no transparent color and send it to the
            (0,0) coordinates of the background image.
                GpBitUDBlt(NULL,&gpDraw[nflip],  0, 0, 80, 150, (unsigned char*)img_char, 0, 0, 80, 150);
                // Perform the top and bottom inversion of the character sprite and output it on the background
                   image of the back surface
              GpTransUDBlt (NULL, &gpDraw[nflip], pos_x, pos_y, 80, 150, (unsigned char*)img_char,
                                    0, 0, 80, 150, 0xef);
            }
            GpSurfaceFlip(&gpDraw[nflip++]);
            nflip %= 2;

            n_tick = GpTickCountGet();
            while ( ( GpTickCountGet() - n_tick ) < 10)
```

<List 5-2> Example Source of Sprite Output (II)

You can perform the left and right inversion of a sprite using the GpTransLRBlt() function. As a result of the sprite inverting, you can get the following output.



<Figure 5-4> Sprite Output Window (II)

The GpBitLRBlt() and GpTransLRBlt() functions have been applied to the left picture and the GpBitUDBlt() and GpTransUDBlt() functions to the right picture. The usage of these functions is almost identical to the one of the GpBitBlt() and GpTransBlt() functions. The only difference is that the sprite output on the display shows the left and right and the top and bottom inversion. You can take the advantage of these functions to generate a wide variety of output with small number of image data.

7. Sound System

7-1 Overview

The GP32 sound system has support for creating PCM sound output and provides effect sound output capability. To allow PCM sound that is effect sound despite short output time to output diverse effect sound, the GP32 system also supports WAV file for Windows. This means the PCM sound can mix and output up to 4 effect sounds concurrently in real time. The GP32 sound supports 8- and 16-bit audio data in stereo, with sample rates of 11kHz, 22kHz and 44kHz. The sound quality is much superior to the one that any other game handheld can support. GP32 also distinguishes itself by an earphone port to enable high-quality sound.

7-2 PCM Sound Output

This is an example of PCM(Pulse Coded Modulation) sound output.

```
#include "gpdef.h"
#include "gpstdlib.h"
#include "gpgraphic.h"
#include "gpmm.h"
#include "gpfont.h"
#include "gpmain.h"
GPDRAWSURFACE gpDraw[2];
int nflip;
extern const unsigned char wav_data[21170];     //wave data 16bit mono 11160Hz
void GpMain(void)
{
        unsigned char keydata;
        unsigned int n_tick;
        int flag_play,i;
        for (i=0; i<2; i++)
        {
                GpLcdSurfaceGet(&gpDraw[i],i);
                        return;
                GpRectFill(NULL, &gpDraw[i], 0, 0, gpDraw[i].buf_w, gpDraw[i].buf_h, 0xff);

        }
        GpSurfaceSet(&gpDraw[0]);

        nflip = 1;
        GpPcmInit(PCM_M11,PCM_16BIT);     //pcm module, initialize sample rate = 11160Hz, mono (flag 0)
        flag_play = 1;
        GpPcmPlay((unsigned short*)wav_data,sizeof(wav_data), 0);       //This function outputs pcm data.
        while(1)
        {
                keydata = GpKeyGet();
                if (keydata == GPC_VK_ENTER)
                {
                        if (flag_play == 1)//Suspend ongoing output
                        {
                                flag_play = 0;
                                GpPcmStop();
                        }
                        else    //Resume the suspended output
                        {
                                flag_play = 1;
                                GpPcmPlay((unsigned short *)wav_data, sizeof(wav_data), 0);
                        }
                }
                GpRectFill(NULL, &gpDraw[nflip], 0, 0, 320, 240, 0xff);
                if (flag_play)
                        GpTextOut(NULL, &gpDraw[nflip], 20, 100, (char*)" Output PCM sound ", 0x0);

                else
                        GpTextOut(NULL, &gpDraw[nflip], 20, 100, (char*)" Stop PCM sound ", 0x0);

                GpSurfaceFlip(&gpDraw[nflip++]);
                nflip %= 2;
                n_tick = GpTickCountGet();
                while ( (GpTickCountGet() - n_tick) < 400)              /* 400msec delay */
                        ;
        }
```

<List 6-2> Example Source of PCM Sound Effect Output

PCM sound is one of the most commonly used sound effect data on the PC. The following 3 APIs must be used in order to output PCM sound data on GP32. First, GpPcmInit() function initializes the sound that you are going to output. Second, GpPcmPlay() outputs the actual sound. Third, GpPcmStop() stops the PCM sound currently being played.

Firstly, the variable to refer to the actual PCM sound data is declared in Line 9 of the List.

```
9:    extern const unsigned char wav_data[21170];    //wave data 16bit mono 11160Hz
```

The GpPcmInit() function configures the environment where the PCM sound will be played on GP32. It sets the sample rate and mono/stereo of the PCM sound you want to output

```
25:        GpPcmInit(PCM_M11,PCM_16BIT);      //pcm module, initialize sample rate = 11160Hz, mono (flag 0)
```

The function is used in Line 25 to set PCM sound with the sample rate of 11160Hz/sec and at mono sound.

The actual sound output is carried out in Lines 27 and 41.

```
27:        GpPcmPlay((unsigned short*)wav_data,sizeof(wav_data), 0);       //Output pcm data
41:                       GpPcmPlay((unsigned short *)wav_data, sizeof(wav_data), 0);
```

This is a detailed explanation of GpPcmPlay().

```
int GpPcmInit(PCM_SR sr, PCM_BIT bit_count);
/*
        <arg 1> PCM_SR sr
                Set sample rate
                PCM_M11,       Mono   11.025KHz
                PCM_S11,       Stereo 11.025KHz
                PCM_M22,       Mono   22.050KHz
                PCM_S22,       Stereo 22.050KHz
                PCM_M44,       Mono   44.100KHz
                PCM_S44,       Stereo 44.100KHz
        <arg 2> PCM_BIT bit_count
                PCM_8BIT,      If  8-bit sound,
                PCM_16BIT      If 16-bit sound,
*/
```

Once you set the output sound as above, the actual PVM sound will be output using GpPcmPlay().

```
/* Function to output actual PCM sound */
int GpPcmPlay(unsigned short * src, int size, int repeatflag);
/*
        <arg 1> unsigned short * src
                Address of actual PCM sound data
        <arg 2> int size
                Length of sound data above
        <arg 3> int repeatflag
                Value whether to repeat or not
                0 is for playing once.
                1 is for repeat.
*/
```

Below is the explanation of GpPcmStop() which stops the PCM sound currently being played.

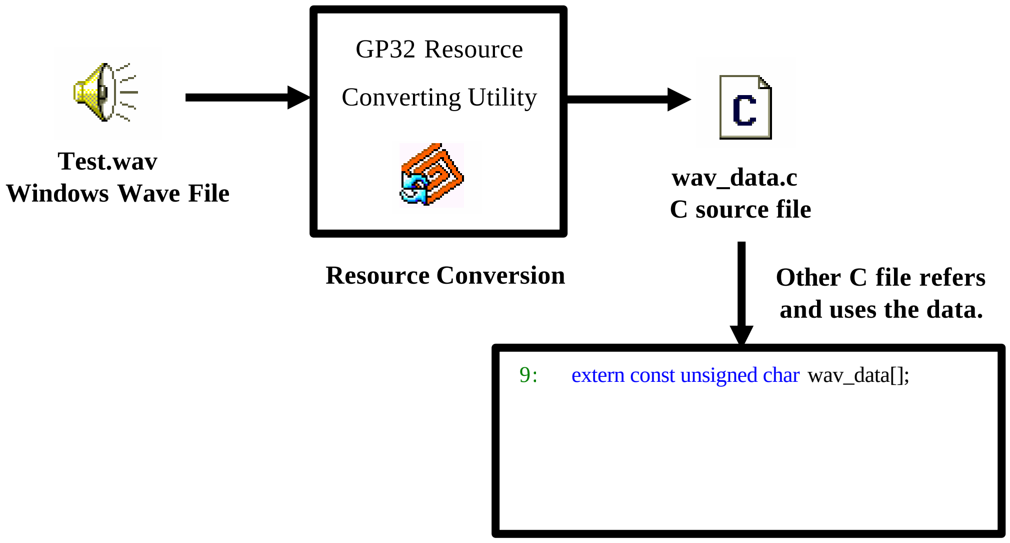
```
/* Function to stop PCM sound output */
void GpPcmStop(void);
/* No input and output values. All sounds are stopped. */
```

PCM sound is also resource data, thus it must be converted in order to use it on GP32 using the

GP32 Resource Converting Utility.

The actual program outputs referring to the converted data.

(See GP32 Dev Utility for more details.)



```
9:     extern const unsigned char wav_data[];
```

<Figure 6-2> Process of Converting PCM Sound Effect

7.3 PCM Sound Mixing

This is an example of mixing four PCM sounds.

```
#include "gpdef.h"
#include "gpstdlib.h"
#include "gpgraphic.h"
#include "gpmm.h"
#include "gpfont.h"
#include "gpmain.h"
GPDRAWSURFACE gpDraw;
extern const unsigned char wav_data1[21170];    //wave data 16bit mono 11160Hz
extern const unsigned char wav_data2[10498];
extern const unsigned char wav_data3[3362];
extern const unsigned char wav_data4[2318];
void GpMain(void)
{
        unsigned char keydata;
        unsigned int n_tick;
        int flag_play,i;

        GpLcdSurfaceGet(&gpDraw, 0);

        GpRectFill(NULL, &gpDraw, 0, 0, gpDraw.buf_w, gpDraw.buf_h, 0xff);
        GpSurfaceSet(&gpDraw);

        GpPcmInit(PCM_M11,PCM_16BIT);       //pcm module initialize, sample rate = 11160Hz, mono (flag 0)
        flag_play = 1;
        GpPcmPlay((unsigned short*)wav_data1,sizeof(wav_data1), 1);     //wav_data1 을 반복해서 출력
        GpPcmPlay((unsigned short *)wav_data2, sizeof(wav_data2), 1);   //wav_data2 를 반복해서 출력
        GpPcmPlay((unsigned short *)wav_data3, sizeof(wav_data3), 1);   //wav_data3 를 반복해서 출력
        GpPcmPlay((unsigned short *)wav_data4, sizeof(wav_data4), 1); //wav_data4 를 반복해서 출력
        /* 5sec 동안 출력 */
        /*Output for 5sec*/
        n_tick = GetTickCount();
        while ( (GetTickCount() - n_tick) < 5000)
               ;
        GpPcmStop();
        GpTextOut(NULL, &gpDraw, 10, 60,
               (unsigned char*)"F1 key : wav 1\nF2 key : wav 2\nF3 key : wav 3\nEnter key : wav 4", 0x0);
        while(1)
        {
               keydata = GpKeyGet();
               //function key 가 변하지 않으면 입력받은 키를 무시(GPC_VK_NONE)
               If function key is not changed, ignore the input key data
               if ((GpKeyChanged() & 0xff ) == 0)
                      keydata = GPC_VK_NONE;
               if (keydata == GPC_VK_F1)
                      GpPcmPlay((unsigned short *)wav_data1,sizeof(wav_data1), 0);    //wav_data1 을 1 회 출력
                      Output wav_data1 once
               else if (keydata == GPC_VK_F2)
                      GpPcmPlay((unsigned short *)wav_data2, sizeof(wav_data2), 0);   //wav_data2 를 1 회 출력
                      Output wav_data2 once
               else if (keydata == GP C_VK_F3)
                      GpPcmPlay((unsigned short *)wav_data3, sizeof(wav_data3), 0);   //wav_data3 를 1 회 출력
                      Output wav_data3 once
               else if (keydata == GPC_VK_ENTER)
                      GpPcmPlay((unsigned short *)wav_data4, sizeof(wav_data4), 0); //wav_data4 를 1 회 출력
                      Output wav_data4 once.

               /* 400msec 동안 지연 */
               /*400msec of delay*/
               n_tick = GpTickCountGet();
               while ( (GpTickCountGet() - n_tick) < 400)
                      ;
        }
}
```

<List 6-3> Example Source of PCM Sound Mixing

Execute the program and 4 PCM sounds will be mixed and output for 5 seconds. After 5 seconds, a message about each key input and OCM sound will appear on the screen and all the sounds will be stopped.
Sound output corresponds to each key input.
Up to 4 PCM sounds can be mixed within the GP32 system. Thus, it would be the same for the programmer whether 1 or more sounds are output.

8. Appendix

8.1 Getting Started with GPSDK bv20

STEP1. Check the directory structure.

gpsdk_bv10\gpinclude ➔ This directory contains the GPSDK header files.

gpsdk_bv10\gplib ➔ This directory contains the GPSDK library files.

gpsdk_bv10\target ➔ This directory contains the GPSDK Start-Up codes and the portable option files.

gpsdk_bv10\misc ➔ This directory contains many other implementation source codes.

gpsdk_bv10\util ➔ This directory contains the GP32 Development Utility BV1.0 version executable files, the standard palette files, pllset.exe files and etc.

STEP 2. Check target\inival_port.h (Here you see the actual source code.)

```
/*********************************************************************/
//at loading time, thread stack define -- implemented in gpstart.c
#define GPMAIN_STACK_SIZE          (100<<10)       //100KB  -- access code = 0
#define NET_STACK_SIZE             (64<<10)     //64KB   -- access code = 1
#define USER_STACK_SIZE            (4 << 10)    //4KB    -- access code = 2
/*********************************************************************/

/***************************************************************
* Heap Management Library Attach                               *
***************************************************************/
#define USE_GP_MEM          1          // If you don't use gpmem.alf, change USE_GP_MEM to 0

/***************************************************************
* Button Checking Loop count                                *
***************************************************************/
#define KEYPOLLING_NUM   20         // You can change polling number,
                                      but the valus must be as small as possible.

/***************************************************************
* Processor Clock speed                                        *
***************************************************************/
#define DEFAULT_MCLK  67800000
#define CHANGE_MCLK         0          // If the CHANGE_MCLK is zero,
                                         the clock speed of process is 40MHz
#if CHANGE_MCLK                        // If the CHANGE_MCLK is non-zero, select CLOCKSPEED
      #define YOUR_SELECT_CLK          0
      #if (YOUR_SELECT_CLK == 0)
            #define CLK_SPEED
            #define DIV_FACTOR
            #define CLK_MODE
      #elif (YOUR_SELECT_CLK == 1 )
      #else
      #endif
#endif /*CHANGE_MCLK*/
```

```
#endif /*__initval_port_h__*/
```

STEP 3. Check target\gpfont_port.h (Here you see the actual source code.)

※ You must replace the content of a text file exported from the GP32 font utility (resource data array) with the content of the gpfontres.dat in order to use other resource instead of the GPSDK font.

```
#ifndef __GPFONT_PORT_H__
#define   __GPFONT_PORT_H__
/* Input information about the system font here. If you want to replace the font resource
      (gpfontres.dat), this is the right place you can revise the relevant information.
*****************************************************************************************/
#define KORFONT_W       12   /* pixel ← Korean font width
#define KORFONT_H       12   /* pixel ← Korean font height
#define ENGFONT_W       8    /* pixel ← English font width
#define ENGFONT_H            16   /* pixel ← English font height
#define  FONT_CHARGAP        4    /* percentage ← Character pitch (See font library)
#define FONT_LINEGAP         4    /* percentage ← Line Spacing (See font library)
#endif
```

STEP 4. Check target\gpstart.c

```
#include "gpdef.h"
#include "gpstdlib.h"
#include "gpmain.h"
#include "gpfont.h"
#include "gpfont_port.h"
#include "gpfontres.dat"

#include "initval_port.h"

#ifdef USE_GP_MEM
#include "gpmem.h"
#endif

unsigned int HEAPSTART;
unsigned int HEAPEND;

void InitializeFont(void);
extern void GpKeyPollingTimeSet(int loop_cnt);
void Main(int arg_len, char * arg_v)
{

      GM_HEAP_DEF gm_heap_def;

#if CHANGE_MCLK  //If CHANGED_MCLK is 1,
      GpClockSpeedChange(CLK_SPEED, DIV_FACTOR, CLK_MODE);
#endif

      _gp_sdk_init();
      //keyboard polling count setting
```

```
	GpKeyPollingTimeSet(KEYPOLLING_NUM);

#ifdef USE_GP_MEM
	gm_heap_def.heapstart = (void*)(HEAPSTART);
	gm_heap_def.heapend = (void *)(HEAPEND & ~3); //<== BOOTROM SPECTIFIC
	gm_heap_init(&gm_heap_def);

	gp_mem_func.malloc = gm_malloc;
	gp_mem_func.zimalloc = gm_zi_malloc;
	gp_mem_func.calloc = gm_calloc;
	gp_mem_func.free = gm_free;
	gp_mem_func.availablemem = gm_availablesize;
	gp_mem_func.malloc_ex = gm_malloc_ex;
	gp_mem_func.free_ex = gm_free_ex;
	gp_mem_func.make_mem_partition = gm_make_mem_part;

	gp_str_func.memset = gm_memset;
	gp_str_func.memcpy = gm_memcpy;
	gp_str_func.strcpy = gm_strcpy;
	gp_str_func.strncpy = gm_strncpy;
	gp_str_func.strcat = gm_strcat;
	gp_str_func.strncat = gm_strncat;
	gp_str_func.lstrlen = gm_lstrlen;
	gp_str_func.sprintf = gm_sprintf;
	gp_str_func.uppercase = gm_uppercase;
	gp_str_func.lowercase = gm_lowercase;
	gp_str_func.compare = gm_compare;
	gp_str_func.trim_right = gm_trim_right;
#endif /*USE_GP_MEM*/

	//Font initialize
	InitializeFont();		//font initialization

	GpKernelInitialize();
	GpKernelStart();
	GpAppExit();
	while(1);
}

void InitializeFont(void)
{
	BGFONTINFO mInfo;
	mInfo.kor_w = KORFONT_W;
	mInfo.kor_h = KORFONT_H;
	mInfo.eng_w = ENGFONT_W;
	mInfo.eng_h = ENGFONT_H;
	mInfo.chargap = FONT_CHARGAP;
	mInfo.linegap = FONT_LINEGAP;
	GpFontInit(&mInfo);
	GpFontResSet((unsigned char*)fontresKor, (unsigned char*)fontresEng);
}

int GpPredefinedStackGet(H_THREAD th)
{
	switch ( th )
	{
	case H_THREAD_GPMAIN:
```

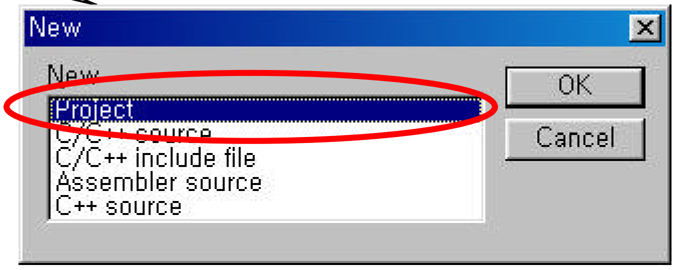
```
            return GPMAIN_STACK_SIZE;
        case H_THREAD_NET:
            return NET_STACK_SIZE;
        case H_THREAD_TMR0:
        case H_THREAD_TMR1:
        case H_THREAD_TMR2:
        case H_THREAD_TMR3:
            return USER_STACK_SIZE;
        default:
            return 0;
        }
}
```

STEP 5. Create a project in the ARM Project Manager

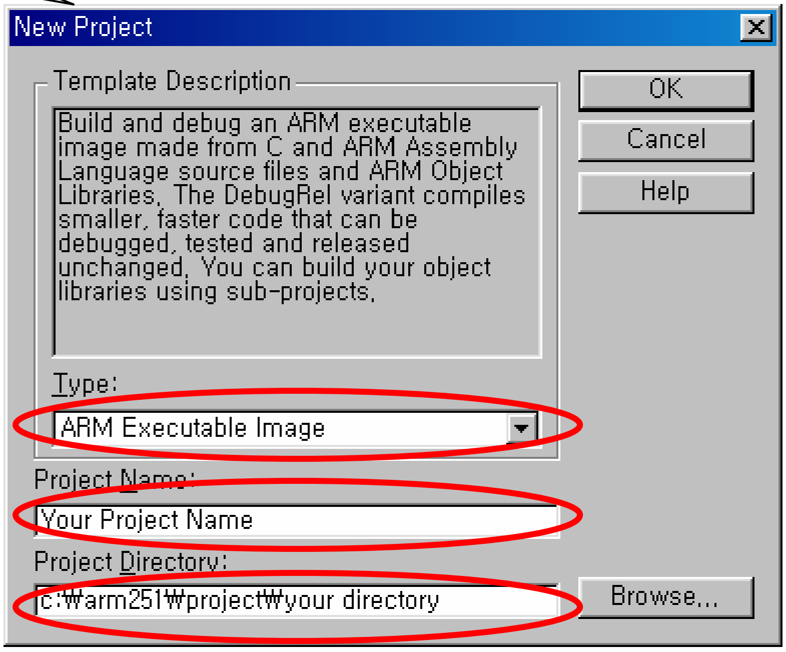
➔ You shall create a directory where you will place a project and copy the \target folder to the directory.

➔ The next steps are as follows. You only have to build after following steps.

Step 1



Step 2

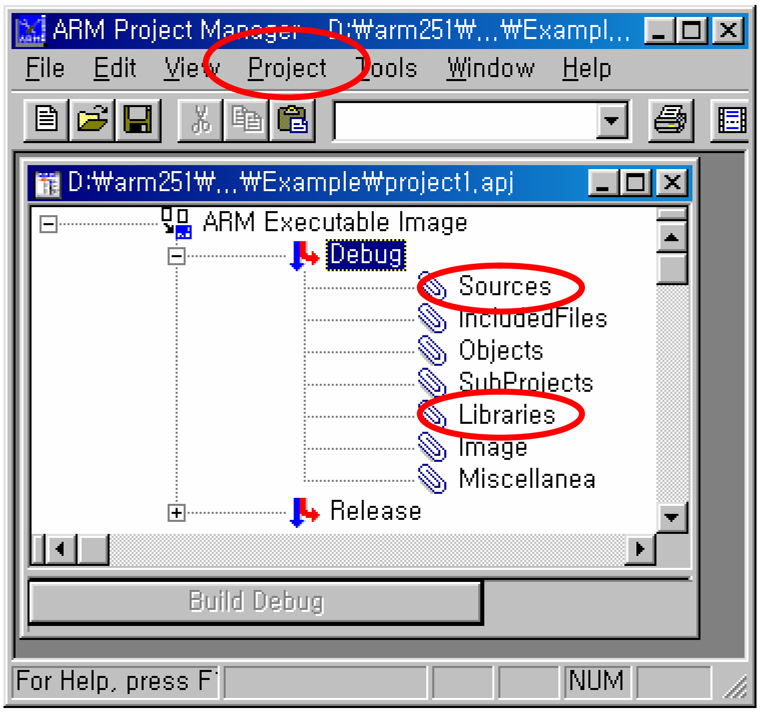


Build and debug an ARM executable image made from C and ARM Assembly Language source files and ARM Object Libraries. The DebugRel variant compiles smaller, faster code that can be debugged, tested and released unchanged. You can build your object libraries using sub-projects.

Type: ARM Executable Image

Project Name: Your Project Name

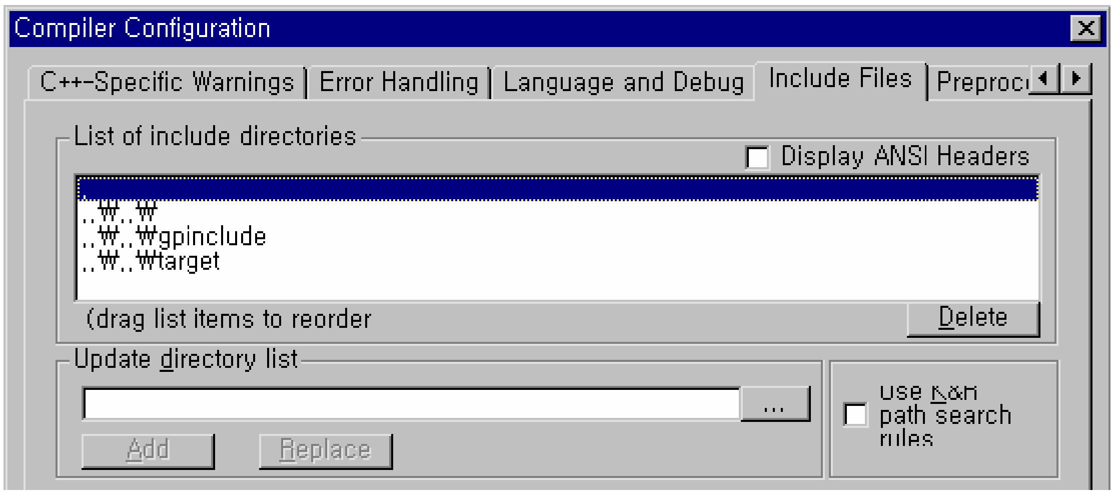
Project Directory: c:₩arm251₩project₩your directory



Step 3

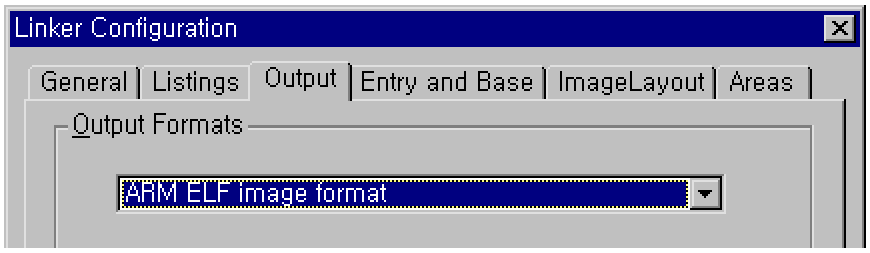
(Menu Project → Tool Configuration for "Debug" →armlink)

1. In the above figure, go to Project and select the Add Files to Project menu.
2. Add the \target\init.s, \target\gpstart.c files.
3. Add the \gplib\ *.alf file (Refer to the Note 1).
4. Configure the Compile and Link options as shown in the picture in Step 4.

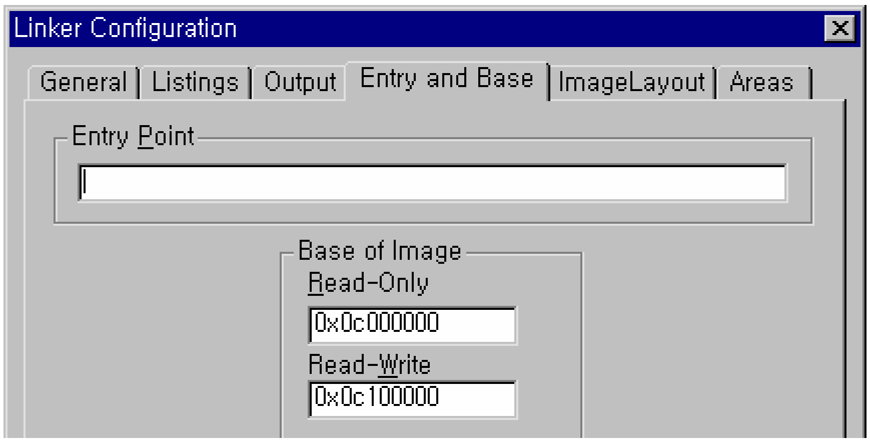


Step 4 (armcc options)

1. Select the Includes Files tab.
2. Click on the … button in the update directory list window.
3. Select the \gpinclude folder and press Add.
4. Enter the ".\target" in the update direct list and press Add.
5. Enter the "." in the update direct list and press Add.
6. Press the Enter key.

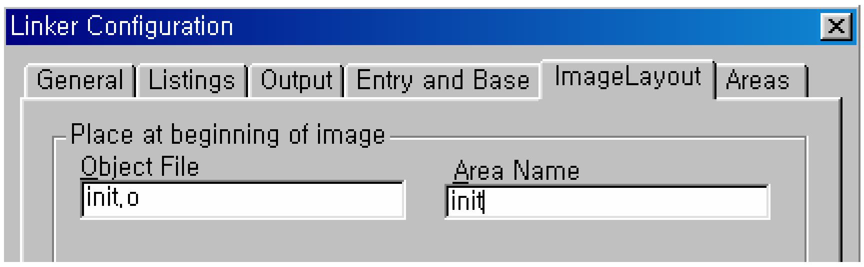


Output Formats: ARM ELF image format



Entry Point:

Base of Image
Read-Only: 0x0c000000
Read-Write: 0x0c100000



Place at beginning of image
Object File: init.o
Area Name: init

Step 5 (armlink option)

1. Select the ARM ELF image format from the Output tab. (See the Note 2)
2. Fill the Read-Only blank in the Entry and Base tab with 0x0c000000. (See the note 3 regarding the Read-Write.)
3. Enter the init.o in the Object File and init in the Area Name in the ImageLayout tab and press the Enter key.

Note1> The alf files are listed as below.

Gpstdlib.alf	➔ Required.
Gpgraphic.alf	➔ Required
Gpgraphic8.alf	➔Optional. This file is required when the API relevant to 8bit graphic is used in the application program
Gpgraphic16.alf	➔ Optional. This file is required when the API relevant to 16bit graphic is used in the application program
Gpfont.alf	➔ Required
Gpfont8.alf	➔Optional. This file is required when the API relevant to 8bit graphic is used in the application program
Gpfont16.alf	➔Optional. This file is required when the API relevant to 16bit graphic is used in the application program
Gpsound.alf	➔ Optional. This file is required when the API relevant to sound is used in the application program.
Gpstdio.alf	➔ Optional. This file is required when the file I/O API is used in the application program.
Gpg_ex01.alf	➔ Optional. This file is required when the API relevant to graphic expansion is used.
Gpmem.alf	➔ Optional. This file is required when the API relevant to Heap and String available for use in the GPSDK BV20 is used in the application program.
Gpnet.alf	➔ Optional. This file is required when the TCP/IP socket API is used in the application program.
init.o	➔ Required. (This file is located in the gplib folder.)

Note 2> The output file of the ARM SDT 2.5x

➔ Select the ARM ELF image format in the Debug mode.

➔ Select the ARM ELF image format” in the Release mode.

Note 3> The Read-Write address option (Link Option)

➔ This option varies by the code size of the application program. As the GP32 adopted the 8mb of SDRAM, the size of address should be optimized to best reserve the Heap area and also should be bigger than the code size. The last two bytes should be terminated with 0.

8-2 The file format used by GP32

GPG File

GP32 Graphic File

0 ~ 3	: File ID. ‘gpg ’ (4byte)
4 ~ 7	: Date size = file size – 8 (4byte)
8 ~	: Data

SEF File

GP32 PCM File

0 ~ 3	: File ID. ‘sef ’ (4byte)	← file header chunk
4 ~ 7	: Data size = file size – 8 (4byte)	
8 ~	: Data	← data chunk

GFT File

GP32 Font File

0 ~ 3	: File ID. ‘gft ’ (4byte)	← file header chunk
4 ~ 7	: Data size = file size – 8 (4byte)	
8 ~	: Data	← data chunk

GXF File

GP32 Application Program File

0 ~ 3	: File ID. ‘gxf ’ (4byte)	← file header chunk
4 ~ 7	: file size – 8 (4byte)	
8 ~ 11	: info Header Size – (4byte)	← info header chunk
12 ~ 12	: icon image flag (if 1, the icon image exists.) (1byte)	
13 ~ 13	: the length of application program title= t_len (1byte)	

14 ~ 13 + t_len : application program title(t_len byte)

14 + t_len ~ 13 + t_len + 256 : icon image data (256 byte)

270 + t_len ~ 273 + t_len : axf (ARM excutable file) data size (4byte) ← data chunk

274 + t_len ~ : axf data