GP2X Hardware Library

Reference Document - version 0.01

a quick documentation

F-Cycles / Quebarium

Table of Contents

1. Overview....3
2. Limitation....3
3. Getting started....3
4. Tools....4
5. Initialization / Free....5
6. Time....5
7. Input....6
8. Display....7
9. RGB....8
10. RGB Debug Helper....8
11. Cursor....9
12. Tiny Menu....9
13. Pixel Data Type....9
14. YUV....10
15. OSD....10
16. SPU....11
17. Hardware Blitters....11
18. Color Adjustments....11
19. Dual Core Programming....12
20. Miscellaneous....12
21. TV Close Caption....12
22. Clock & Power Management....13
23. Direct Memory and I/O Access....13
24. Profiler....13
25. BMP Dump....13
26. Sound....14
27. Samples....14
28. References....16

1. Overview

We recommend to look and read the overview slide document unless you are very familiar with GP2X architecture. This document is written as a reference document and we assume that you have a good programming experience in developing multimedia software.

This library and sample applications has been developed using a version of Dev-C++ provided by GPH.

We use the prefix 'qbr' (for Quebarium) across the library to avoid name collision with other library.

For more information about hardware feature, please look in the reference section and refer to the machine specification documents listed.

2. Limitation

This library is currently limited to 'C' language due to the current dual core implementation.
This library has been developed and tested using GPH Firmware (v2, v3 and v4), it may not work with other firmware like Open2X.

3. Getting started

To create a new project:

We recommend to copy the sample found in TinyMenu into a new directory, edit the .DEV file in your favorite text editor and make sure the path to lib files are corrects.

Functions and macros use in the library are listed in qbr_lowLevelLib.h. Additionnal functions can be found in qbr_lowLevelDbg.h and qbr_basicTools.h. The use of Close-Caption needs qbr_LowLevelcc.h.

4. Tools

bin2c

Create a source file (.h) which can be use within your project using any binary file.
Useful to include all resources internally, so your project will be only one .gpe file.
You can shrink your .gpe file with resources included using a packer.

Example: bin2c.exe image.jpg image.h

bmp2gp2x

Convert image (.bmp file format 24-bit) to GP2X data format:
- Index (Cursor: 2-bit, RGB: 4/8-bit, OSD: 8-bit)
- SPU (compressed data)
- RGB (24 / 16 bits)
- YUV (YUY2 data type)
- Randomize Coordinate 8-bit: X, Y (for morphing effect)

Utility which parse a script file (filename.txt) and using the source image (filename.bmp) produce a binary and map file (filename.bin, filename.map) or source file (filename.h) to be use in your GP2X application.

Example:

1. RGB 16-bit:

```
        FILE_OUTPUT = BINARY
OUTPUT_FORMAT = RGB888
ADD("Tarsier",0,0,720,960,1)
```

2. SPU:

```
FILE_OUTPUT = SOURCE
OUTPUT_FORMAT = SPU
        // if SPU need to be interlaced, we will specify OUTPUT_FORMAT = SPU_INTER
COLOR_TO_INDEX(0) = #5D3E28
COLOR_TO_INDEX(1) = #8F4129
COLOR_TO_INDEX(2) = #A68B1B
COLOR_TO_INDEX(3) = #D2CBAA
ADD("Nausicaa",0,0,320,240,1)
```

3. OSD:

```
FILE_OUTPUT = SOURCE
OUTPUT_FORMAT = IRAW
BIT_PER_PIXEL = 8
COLOR_TO_INDEX(0) = #000000
COLOR_TO_INDEX(1) = #000008
COLOR_TO_INDEX(2) = #000010
COLOR_TO_INDEX(3) = #000018
...
COLOR_TO_INDEX(254) = #F9EAED
COLOR_TO_INDEX(255) = #F7FFF7
ADD("Buran",0,0,720,576,1)
```

4. YUV:

```
FILE_OUTPUT = SOURCE
OUTPUT_FORMAT = YUY2
ADD("Kamakura",0,0,320,240,1)
```

pal2script

Utility to convert index palette code listing to format useful to bmp2gp2x.

Example: pal2script.exe c64Fli.pal
Produce a file: c64fli.pal.txt

See: OSD sample source code for more info.

scrStats

Utility that scan C/C++ source files and provide you some statistics like the number of lines of code written. This utility could be use to blog about your project and how it progress over the weeks.

5. Initialization / Free

Initialization and release of resources within the library are achieve with these functions.

```
QBR_RESULT      QBR_InitLowLevelAPI( char *strProjectName );
QBR_RESULT      QBR_FreeLowLevelAPI( BOOL bLaunchGP2XMenu );
```

6. Time

Many functions are provided related to time. Animation should always be render base on the current time in order to remain accurate using QBR_GetTime or QBR_GetFrameTime. Waiting for vertical sync is achieve via QBR_WaitVSync.

Waiting for task completion can be achieve using QBR_WaitHwBlitter and QBR_WaitArm940T (these are 2 blocking calls).

```
QBR_RESULT  QBR_ResetFrameTime( void );
QBR_RESULT  QBR_GetFrameTime( unsigned long *ulTime );
QBR_RESULT  QBR_GetFrameCount( unsigned long *ulCount );
QBR_RESULT  QBR_GetTime( unsigned long *ulTime );
QBR_RESULT  QBR_DelayMS( unsigned long ulDelay );
QBR_RESULT  QBR_WaitHSync( void );
QBR_RESULT  QBR_WaitVSync( void );
QBR_RESULT  QBR_WaitHwBlitter( BOOL bRotateBlitter );
QBR_RESULT  QBR_WaitArm940T( void );
```

7. Input

GP2X Input:

To read the GP2X control state, you should use QBR_GetGP2XInputState combine with the define QBR_GP2X_INPUT_MASK_DPAD_UP to QBR_GP2X_INPUT_MASK_EXIT.

The QBR_GP2X_INPUT_MASK_EXIT is a virtual button which is map to ESCAPE on PC and LEFT + RIGHT trigger on GP2X. Be aware that QBR_GP2X_INPUT_MASK_DPAD_FIRE is not available on model F200.

```
QBR_RESULT      QBR_GetGP2XInputState( unsigned long *ulInputState );
```

Touch Screen Input:

The touch screen is only available on F200. We recommend to take a look to TouchScreen and LibTest sample.

```
QBR_RESULT      QBR_GetTouchScreen( unsigned short *x, unsigned short *y, unsigned short *pressure );
```

HID Input:

You can access and detect USB HID devices like gamepad, keyboard and mouse. First, you need to configure your GP2X to enable these features. Refer to LibTest sample code.

```
QBR_RESULT      QBR_GetHIDEvent( unsigned short usDeviceIndex, unsigned short *usType,
                                 unsigned short *usCode, unsigned int *unValue );
QBR_RESULT      QBR_HIDDetection( BOOL bOpenDevices );
```

8. Display

There are five different native display output available on the GP2X:

- LCD 320 x 240 @ ? FPS
- TV NTSC interlaced 720x480 @ ~30 FPS per field
- TV NTSC progressive 720x240 @ ~60 FPS
- TV PAL interlaced 720x576 @ ~25 FPS per field
- TV PAL progressive 720x288 @ ~50 FPS

From that, we added different RGB scaler values to create lower resolution. All TV resolution listed are borderless.

Display Mode List:
==========

- QBR_DISP_LCD_320x240	0	// ** Default LCD (and boot) mode on GP2X Kernel/Menu v2.1.1 **
- QBR_DISP_LCD_160x240	1	// LCD Pixel size 2x1.
- QBR_DISP_LCD_160x120	2	// LCD Pixel size 2x2.
- QBR_DISP_LCD_80x120	3	// LCD Pixel size 4x2.
- QBR_DISP_LCD_80x60	4	// LCD Pixel size 4x4.
- QBR_DISP_TV_NTSC_320x240	5	// ** Default TV NTSC mode on GP2X Menu v2.1.1 (lines missing). **
- QBR_DISP_TV_NTSC_720x480	6	// 4:3 = 1.125, Effective = ~1.22 (16:9 = 0.84, Effective = ~0.92)
- QBR_DISP_TV_NTSC_720x240	7	// 4:3 = 2.25, Effective = ~2.45 (16:9 = 1.69, Effective = ~1.84)
- QBR_DISP_TV_NTSC_584x480	8	// 4:3 Effective = ~1.0
- QBR_DISP_TV_NTSC_512x240	9	// Can use AND-Operator 0x1FF.
- QBR_DISP_TV_NTSC_392x240	10	// 16:9 Effective = ~1.0
- QBR_DISP_TV_NTSC_360x480	11	// Pixel size: 2x1
- QBR_DISP_TV_NTSC_360x240	12	// Pixel size: 2x2
- QBR_DISP_TV_NTSC_296x240	13	// 4:3 Effective = ~1.0
- QBR_DISP_TV_NTSC_256x240	14	// Can use AND-Operator 0x0FF.
- QBR_DISP_TV_NTSC_184x240	15	// Pixel size: 4x2 (round up to factor of 8)
- QBR_DISP_TV_NTSC_184x120	16	// Pixel size: 4x4 (idem)
- QBR_DISP_TV_NTSC_96x120	17	// Pixel size: 8x4 (idem)
- QBR_DISP_TV_NTSC_96x60	18	// Pixel size: 8x8 (idem)
- QBR_DISP_TV_PAL_320x240	19	// ** Default TV Pal mode on GP2X Menu v2.1.1 **
- QBR_DISP_TV_PAL_720x576	20	// 4:3 = 0.94, Effective = ~1.0 (16:9 = 0.70, Effective = ~0.76)
- QBR_DISP_TV_PAL_720x288	21	// 4:3 = 1.88, Effective = ~2.0 (16:9 = 1.41, Effective = ~1.51)
- QBR_DISP_TV_PAL_704x576	22	// 4:3 Effective = ~1.0
- QBR_DISP_TV_PAL_512x288	23	// Can use AND-Operator 0x1FF.
- QBR_DISP_TV_PAL_392x288	24	// 16:9 Effective = ~1.0
- QBR_DISP_TV_PAL_360x576	25	// Pixel size: 2x1
- QBR_DISP_TV_PAL_360x288	26	// Pixel size: 2x2 (4:3 Effective = ~1.0)
- QBR_DISP_TV_PAL_352x288	27	// 4:3 Effective = ~1.0
- QBR_DISP_TV_PAL_256x288	28	// Can use AND-Operator 0x0FF.
- QBR_DISP_TV_PAL_184x288	29	// Pixel size: 4x2 (round up to factor of 8)
- QBR_DISP_TV_PAL_184x144	30	// Pixel size: 4x4 (idem)
- QBR_DISP_TV_PAL_96x144	31	// Pixel size: 8x4 (idem)
- QBR_DISP_TV_PAL_96x72	32	// Pixel size: 8x8 (idem)

All TV resolution assume interlaced (default) mode.

Note: Higher resolution like 800x600 or 1024x768 could be obtain but they will flicker even more due to the limitation of the S-Video and composite standard.

Progressive Scan

In order to turn off interlace, you need to call the function QBR_SetTvInterlace(FALSE) after setting one of the TV setting. If you which to change output mode after setting the TV to progressive scan, you first need to turn it off prior to change mode. Once done, you may need to call QBR_SetTvInterlace(FALSE) again if you want to use a progressive scan.

```
QBR_RESULT      QBR_GetDisplayMode( QBR_DISPLAY_MODE *qbrDisplayMode, int *nRGBbpp );
QBR_RESULT      QBR_SetDisplayMode( QBR_DISPLAY_MODE qbrDisplayMode, int nRGBbpp );
QBR_RESULT      QBR_SetTvInterlace( BOOL bInterlace );
QBR_RESULT      QBR_GetVideoBasePtr( void **pVideoBasePtr );
QBR_RESULT      QBR_GetDisplaySize( int *nSizeX, int *nSizeY );
```

9. RGB

The RGB plane is under the Cursor plane but above all other (YUV, SPU and OSD). It could be set to 16-bit (5 bit: red and blue, 6 bit: green) or 24-bit (8 bit: red, green and blue). Or it could be set to an index mode using a color palette (4 or 8-bit).

The RGB plane have a 5 rectangle regions features which allow to adjust different regions on screen with different transparency settings. It is possible to have one color reserve for full transparency. Unlike OSD plane, RGB colors do not have any alpha setting.

The start address of the RGB plane should be 4-byte align. The RGB color key is always a 24-bit value even when RGB is set to 16-bit.

```
QBR_RESULT      QBR_SetPalette( unsigned short usIndex, unsigned long *pulPaletteData,
                        unsigned short usCount );
QBR_RESULT      QBR_SetRGBStartAddress( unsigned long ulStartAddress );
QBR_RESULT      QBR_SetRGBRegionColorKey( unsigned long ulColorKeyValue );
QBR_RESULT      QBR_SetRGBRegionOperation( int nRegion, BOOL bEnable, BOOL bActivate,
                        unsigned short usMixMuxOp, unsigned short usAlphaValue, unsigned short usLeft,
                        unsigned short usTop, unsigned short usRight, unsigned short usBottom );
```

See: RGB sample source code for more info.

10. RGB Debug Helper

A 40x25, 16-colors text mode provide an easy way to draw text on RGB plane which can be found in qbr_lowLevelDbg.h. The text will auto-wrap and auto-scroll up as you add text to the screen buffer using QBR_DbgPrint (equivalent to printf). The render of text (in 16 and 24-bit mode) could also be render in 25%, 50% or 75% transparency mixed with the existing RGB.

In 4 or 8-bit palette RGB, you should set the last 16 colors of the palette by using the 16 colors define in qbr_lowLevelDbg.h. Colors and charset is inspired from the Commodore 64 and 128 machine machine from the '80.

```
QBR_RESULT QBR_DbgClrScr( int nBorderColor, int nBackgrndColor );
QBR_RESULT QBR_DbgPrint( char *string, ... );
QBR_RESULT QBR_DbgGotoXY( int nPosX, int nPosY );
QBR_RESULT QBR_DbgColor( int nForegrndColor, int nBackgrndColor );
QBR_RESULT QBR_DbgDrawScr( QBR_DBG_DRAW_TYPE ulDrawType, unsigned long ulStartAddress );
```

See: most sample source code are using these functions to display text.

11. Cursor

The cursor plane is a 32x32 or 64x64 2-bit per pixel sprite. The 2-bit allow you to select between these 4 choices: 2 colors, full transparency and inversion.

```
QBR_RESULT      QBR_SetCursorParams( BOOL bEnable, unsigned short usSize, unsigned long ulBckgrndColor,
                                     unsigned long ulForegrndColor, unsigned long ulStartAddress );
QBR_RESULT      QBR_MoveCursor( unsigned short usPosX, unsigned short usPosY );
```

See: TinyMenu source code which is using Cursor.

12. Tiny Menu

The Tiny Menu offers a way to use the Cursor to display user feedback information on volume control and pause/exit. On the GP2X, there is 2 buttons for volume control (Vol - and Vol +), we thought it would be nice to get a visual feedback when the user press on it to adjust the volume. Tiny Menu offer you a way to display a small progress bar under the word 'Volume' which can vary or be replace by the word 'muted'. We suggest to mute the volume when the user press both button and unmute as soon one of the two is press.

Another intersting feature we thought, would be to be able to pause the title and display a tiny menu in center of the screen which allow the end-user to resume or exit. Another feature (camera) has been added which consist of having the title paused without the tiny menu on screen. Pressing any key should put back the menu on screen.

Tiny Menu offer an easy way to have a pleasant animation available in 2 size (small for LCD and big for TV) to display those information.

See: TinyMenu source code for example.

13. Pixel Data Type

The RGB is often the most common pixel data type. On GP2X, YUV plane use YUY2, OSD use YUVA, while RGB use 16 or 24-bit RGB. We provide in this library a set of functions to convert pixel data type.

```
ULONG       QBR_RGBtoYUY2Color(int nRed, int nGreen, int nBlue );
void        QBR_YUY2toRGBColor(ULONG ulYUY2, int *nRed, int *nGreen, int *nBlue);
ULONG       QBR_Dual16bitRGBtoYUY2Color(unsigned long ulDual16BitRGB);
void        QBR_YUY2ColortoDual16bitRGB( ULONG ulYUY2, unsigned long *ulDual16BitRGB);
ULONG       QBR_RGBAtoYUVAColor(int nRed, int nGreen, int nBlue, int nAlpha);
void        QBR_YUYAtoRGBAColor(ULONG ulYUYA, int *nRed, int *nGreen, int *nBlue, int *nAlpha);
```

14. YUV

The YUV plane is a 16-bit per pixel plane with some hardware feature that allow some hardware horizontal and vertical mirror operations. Pixel are grouped by two which has theirs own individual 8-bit lightness value Y while sharing a U and V value. Stretching and shrinking can be achieve and should be set each time configuring the YUV through QBR_SetYUVVideoPlaneScaler().

```
QBR_RESULT  QBR_SetYUVVideoPlane( BOOL bYUVVideoLayerA, BOOL bEnable,
        unsigned short usLeft, unsigned short usTop,
        unsigned short usRight, unsigned short usBottom,
        unsigned short usSplitPosY, unsigned short usSrcImageWidth,
        unsigned long ulStartAddressTop, BOOL bTopHMirror, BOOL bTopVMirror,
        unsigned long ulStartAddressBottom, BOOL bBottomHMirror, BOOL bBottomVMirror );
QBR_RESULT  QBR_SetYUVVideoPlaneScaler( BOOL bYUVVideoLayerA, BOOL bTopPart,
        unsigned short usHorizontalScale, unsigned short usVerticalScale);
QBR_RESULT  QBR_SetYUVVideoPlanePriority( BOOL bYUVVideoLayerA );
```

See: YUV source code for example.

15. OSD

The OSD plane is a 8-bit per pixel plane with a 24-bit color + 4-bit alpha value for each color entry. One or many OSD regions can be describe, the last header should point to 0xFFFFFFFF for the next OSD Header. Data of OSD would be place right after a 12 bytes OSD Header. When using TV interlaced mode, the ODD and EVEN field must be located into two different address space.

```
QBR_RESULT  QBR_SetOSDPaletteUsingRGB( unsigned short usIndex, unsigned long *pOSDPaletteData,
                unsigned short usCount );
QBR_RESULT  QBR_SetOSDPalette( unsigned short usIndex, unsigned long *pulOSDPaletteData,
                unsigned short usCount );
QBR_RESULT  QBR_SetOSDPlane( BOOL bEnable, unsigned long ulAdrHeaderEvenField,
                unsigned long ulAdrHeaderOddField );
QBR_RESULT  QBR_WriteOSDHeader( unsigned long ulAdrHeader, unsigned long ulAdrNextHeader,
                unsigned short usLeft, unsigned short usTop, unsigned short usRight,
                unsigned short usBottom, unsigned short usBitPerPixel,
                BOOL bRLECompression, BOOL bGlobalAlphaMix, BOOL bAlphaColorKey,
                unsigned short usGlobalAlphaValue, unsigned short usAlphaColorKeyIndex);
```

See: OSD source code for example.

16. SPU

The SPU plane is a 2-bit per pixel plane with a 16 color palette. A 4-bit alpha can be specify for each color. Additional color features can be specify through the Change and Highlight elements. The data must be compressed in a RLE format. We recommend the use of bmp2gp2x in order to convert your bitmap into SPU format. Please note, that in TV Interlaced mode, you will need to provide 2 set of data (one for ODD and one for EVEN lines).

```
QBR_RESULT  QBR_SetSPUColor( unsigned short usIndex, unsigned long ulYCrCbColor );
QBR_RESULT  QBR_SetSPUPlane( BOOL bEnable, BOOL bTopOfOSDPlane, BOOL bHighlight,
                unsigned long ulSPUBaseCmdAdr, unsigned long ulSPUHighlightInfoAdr );
QBR_RESULT  QBR_WriteSPUBaseCmd( unsigned long ulAdrBaseCmd, BOOL bChangeCmd,
                unsigned long ulChangeCmdAdr,  unsigned short usBaseColor, unsigned short usBaseContrast,
                unsigned short usTop, unsigned short usBottom,
                unsigned long ulEvenFieldDataAdr, unsigned long ulOddFieldDataAdr );
QBR_RESULT  QBR_WriteSPUChangeCmd( unsigned long ulAdrChangeCmd, BOOL bLastChangeCmd,
                unsigned short usTop, unsigned short usBottom, unsigned short usChangeInfoCount,
                unsigned short *usStartX, unsigned short *usEndX,
                unsigned short *usChangeColor, unsigned short *usChangeContrast );
QBR_RESULT  QBR_WriteSPUHighlightInfo( unsigned long ulAdrHighlightInfo, unsigned short usLeft,
                unsigned short usTop, unsigned short usRight, unsigned short usBottom,
                unsigned short usHighlightColor, unsigned short usHighlightContrast );
```

See: SPU source code for example.

17. Hardware Blitters

There is 2 hardware blitters which can be use while doing some other tasks with the CPUs.
The address 0 refer to the base address of the video memory.

```
QBR_RESULT  QBR_HwBitBlt( unsigned long ulDestAddress, unsigned long ulDestWidth,
                unsigned short usSizeX, unsigned short usSizeY,
                unsigned long ulSrcAddress, unsigned long ulSrcWidth,
                QBR_HWBITBLP_ROP rasterOperation, unsigned short usTransparencyColor,
                unsigned short usBackgrndColor, unsigned short usForegrndColor );
QBR_RESULT  QBR_HwBitBltRotate( unsigned long ulDestAddress, unsigned long ulDestWidth,
                unsigned short usSizeX, unsigned short usSizeY, unsigned long ulSrcAddress,
                unsigned long ulSrcWidth, QBR_HWBITBLTROT_ROP rotateRasterOperation );
```

See: Blitters source code for example.

18. Color Adjustments

You can modify pixel colors throught a 256 entries table for each color component (R, G and B) via
the gamma functions. Brightness, Contrast, saturation and HUE can be adjust on all YUV color planes: YUV, OSD and SPU.

```
QBR_RESULT  QBR_EnableGamma( BOOL bEnable );
QBR_RESULT  QBR_SetGamma( unsigned short usIndex, unsigned long *pulGammaData,
                        unsigned short usCount );
QBR_RESULT  QBR_SetYUVBrightness( signed char ucBrightness );
QBR_RESULT  QBR_SetYUVContrast( unsigned char ucContrast );
QBR_RESULT  QBR_SetYUVSaturationAndHUE( int nQuadrantIndex, unsigned char ucSaturation,
                        unsigned char ucHUE );
```

See: OSD code for example using QBR_SetYUVSaturationAndHUE.

19. Dual Core Programming

All samples provided with this library are ready for Dual Core programming. The first and master core entry point is in main.c using the usual main function. The second core main entry point and all code should be place in main940t.c.

The binary of your code in this section will be copied in memory elsewhere. For that reason, you cannot call functions unless their source code is copied into this file. Also, the 2nd core cannot do hardware division. We provide two functions in the dual core sample: division and random.
Both core can access the video memory directly.

In order to exchange data between cores, we recommend to use the volatile part of memory. Otherwise, your program may use a cached value.

```
QBR_RESULT  QBR_DualCorePause( BOOL bPause );
QBR_RESULT  QBR_DualCoreSendCmd( unsigned long ulCommand );
QBR_RESULT  QBR_DualCoreStart( unsigned long *ulStartAdrSrc, unsigned long ulSize );
QBR_RESULT      QBR_DualCoreStop( void );
QBR_RESULT      QBR_DualCoreGetSendCmdCount( unsigned long *ulCount );
QBR_RESULT      QBR_DualCoreGetCmdProcessedCount( unsigned long *ulCount );
```

See: DualCore source code for example.

20. Miscellaneous

Various functions can be use in order to know the device state.

```
QBR_RESULT  QBR_GetCurrentTVFieldOutput( unsigned long *ulTVField );
QBR_RESULT      QBR_GetSDState( void );
QBR_RESULT      QBR_GetHardwareModel( unsigned long *ulGp2xModel);
QBR_RESULT      QBR_GetFirmwareVersion( unsigned long *ulVersion );
```

See: LibTest source code for example.

21. TV Close Caption

TV Close Caption should be use with interlaced TV output only. Test using progressive scan has demonstrate to not work on some TV while others were scrolling text irraticaly. Close-Caption code are send two byte at every frame while 2 channels is send on EVEN field and the 2 other on ODD field output.
We think that progressive scan cause the routine running in the TV to never reach some instruction due to the missing field, causing the problem of TV Close Caption while being in progressive scan.

Also, each TV could have is own charset and features support. Some will allow transparency and background color changes, while other will be constrain with solid black background.

```
QBR_RESULT      QBR_CC_SetChannel( int nChannel );
QBR_RESULT      QBR_CC_SetCommand( unsigned short usCommand );
QBR_RESULT      QBR_CC_SetPosition( int nColumn, int nRow, unsigned char ucForegrndColor );
QBR_RESULT      QBR_CC_SetColor( unsigned char ucForegrndColor, unsigned char ucBackgrndColor );
QBR_RESULT      QBR_CC_printf( char *string, ... );
QBR_RESULT      QBR_CC_OutputToTV( void );
```

See: TVCloseCaption source code for example.

22. Clock & Power Management

You can get and change CPU clock speed. OVERCLOCKING AT YOUR OWN RISKS! You can control the battery LED and query the power state (battery or AC adapter).

```
QBR_RESULT      QBR_GetBatteryLEDState( void );
QBR_RESULT      QBR_SetBatteryLEDState( BOOL bLEDStateON );
QBR_RESULT      QBR_GetPowerState( unsigned short *usPowerState ); // <678 Low Batteries,
                        678-745 Medium Batteries, >745 Full Batteries, >870 AC Adapter.
QBR_RESULT      QBR_GetCPUClock( unsigned long *ulCPUClockInMHz );
QBR_RESULT      QBR_SetCPUClock( unsigned long ulCPUClockInMHz ); // We should not call this function
                        while music is playing.
```

See: ClockSpeed source code for example.

23. Direct Memory and I/O Access

You can peek and poke into memory on the MMSP2 and access the CX25874 after switching to TV-Out (in order to power up the chip).

```
QBR_RESULT      QBR_ReadMemRegister( unsigned short usAddress, unsigned short *usValue );
QBR_RESULT      QBR_WriteMemRegister( unsigned short usAddress, unsigned short usValue );
QBR_RESULT      QBR_InCX25874( unsigned char ucAddress, unsigned char *ucValue );
QBR_RESULT      QBR_OutCX25874( unsigned short ucAddress, unsigned short ucValue );
```

See: LibTest source code for example. You can also connect a USB keyboard and use LibTest to read and write live a GP2X.

24. Profiler

Profiling can be made using timer function with basic operations. However, that additional code should be remove in the release version. In order to ease the use of profiling, we provide a set of macros which allow profiling which can be remove at release time by commenting a single define.

See: Profiler source code for example.

25. BMP Dump

This feature is available ONLY on PC as the GP2X mixed video plane in hardware. This feature allow you to save screen shots with a sequence number which can be use to make a movie.

A feature implemented but not tested should allow you to export a time-frame file when running your non-interactive title on GP2X to be use on PC in order to generate the movie with the accurate frame-rate. To use that feature, you need to uncomment _QBR_GP2X_LOWLEVEL_API_FRAMES_TIME in qbr_lowLevelLib.h. Make sure to re-comment that line for final release (as you may not want time-frame information being generated all the time).

```
QBR_RESULT  QBR_SaveOnDisk( BOOL bEnable, char *strFilePath );
```

See: BMPDump source code for example.

26. Sound

There is no implementation currently related with sound system. Althought, we provide one sample (space demo) which use mikmod to play music and sounds. You could find a pre-compiled version of mikmod in sample\space demo.

27. Samples

Included with this library, we provide several code samples where some of them were develop during research of how the features work or study of the performance of the hardware.

- LibTest:

    Test application which display many information about the current hardware state.
    Using a USB Keyboard (you need to enable USB port on GP2X), you can read or write
    memory or I/O. The functions keys (F1 to F12) are assign to some features for
    testing (as well the GP2X keys allowing to test some features without the craddle).

- BatteryTester:

    Test application to study battery charge over time.

- Blitters:

    Test application to show the use of the two hardware blitters: rotary-blitter and
    faster blitter with different operator.

- BMPDump:

    Test application to show on PC how to save render on file.

- ClockSpeed:

    Test application showing under-clocking results.
    OVERCLOCKING AT YOUR OWN RISK.

- DualCpu:

    Test application which process various function on the 2nd core.

- OSD:

    Test application which display a 256 colors image on OSD. Hardware saturation and HUE control is also shown.

- Profiler

    Test application to demonstrate the use of profiler functions comparing sine float software
    function versus the use of a pre-computed table.

- RGB:

  Test application to demonstrate 4 different RGB color mode: 24-bit, 16-bit, 256 and 16 color mode.
  The 256 and 16 colors mode display the same picture using a dithering method combine with a custom palette computed at the beginning.

- SpaceDemo:

  An early benchmark test using the 2 CPU and the hardware blitters. The master and first CPU handle the music/sound effect, sprites and inputs. The second CPU is responsible for computing the background animation (stars and earth).

- SPU:

  Test application to show a use of SPU holding compressed data for a full screen animation on LCD.
  In this example, we have use a dithered picture which cause the RLE compression to be ineffective.
  However, this sample show that complex and long data stream could be use in hardware.
  Also, this sample show the use of highlight region moving on screen.

- TinyMenu:

  Test application to show handling of volume (+, -, mute and unmute) and pausing menu. We recommend to use this sample as the base of your new project. Just copy all the files within another directory, edit the .dev file in your favorite text editor and make sure the reference for source file of this library still match and you are go to start your project!

- TouchScreen:

  Test application which show the detection and use of the touch screen on F200.

- TVCloseCaption:

  Test application which output close caption channel 1.

- TVProgressiveScan:

  Test application which is use to test the TV Scan mode (interlace and progressive).

- YUV:

  Test application which show the use of the 4 YUV regions with flipping feature. The picture and water effect are being achieve in hardware which limit the amount of resources required for this application.

28. References

1. MMSP(tm)2 Data Book MP2520F, Version 1.0
   MagicEyes Digitals Co., Copyright (C) 2001/2/3
   http://mudiweb.com/gp2x/MP2520F_Manual_Eng_V1.0.pdf

2. CX25784/5 Digital Encoder with Standard-Definition TV
   and High-Definition TV Video Output, Conexant(R), August 2004
   http://web.archive.org/web/20061016081745/http://www.conexant.com/servlets/DownloadServlet/DSH-200670-001.pdf?docid=671&revid=1

3. Rlyeh's Minimal Library SDK, Version 0.C
   Copyright(C) 2005-2006
   http://wiki.gp2x.org/wiki/Rlyeh's_Minimal_Library_SDK

4. Closed Caption Disassembly Documentation: Characters
   http://www.theneitherworld.com/mcpoodle/SCC_TOOLS/DOCS/CC_CHARS.HTML