FM4

Graphics Driver Manual for 2D core of S6E2D devices

32-BIT MICROCONTROLLER
FM4 Family

APPLICATION NOTE

Publication Number S6E2DH_AN709-00022 Revision 1.0 Issue Date September 11, 2015

Target products

This application note is described about below products;

(TYPE4-M4)

Series	Product Number (not included Package suffix)
S6E2DH	S6E2DH5G0A, S6E2DH5GAA, S6E2DH5GJA, S6E2DH5J0A, S6E2DH5JAA
S6E2DF	S6E2DF5G0A, S6E2DF5GAA, S6E2DF5GJA, S6E2DF5J0A, S6E2DF5JAA
S6E2D5	S6E2D55G0A, S6E2D55GAA, S6E2D55GJA, S6E2D55J0A, S6E2D55JAA
S6E2D3	S6E2D35G0A, S6E2D35GAA, S6E2D35GJA, S6E2D35J0A, S6E2D35JAA

Table of Contents

Figures

1. 2D Graphics Driver

1.1 About Document

This manual explains the functions of Cypress's 2D Graphics Driver for the S6E2D devices. Please refer to the Delivery Note of your product for devices currently supported with this product.

Introduction

Getting Started

Overview

Tutorial

Glossary

Revision History

2. Introduction

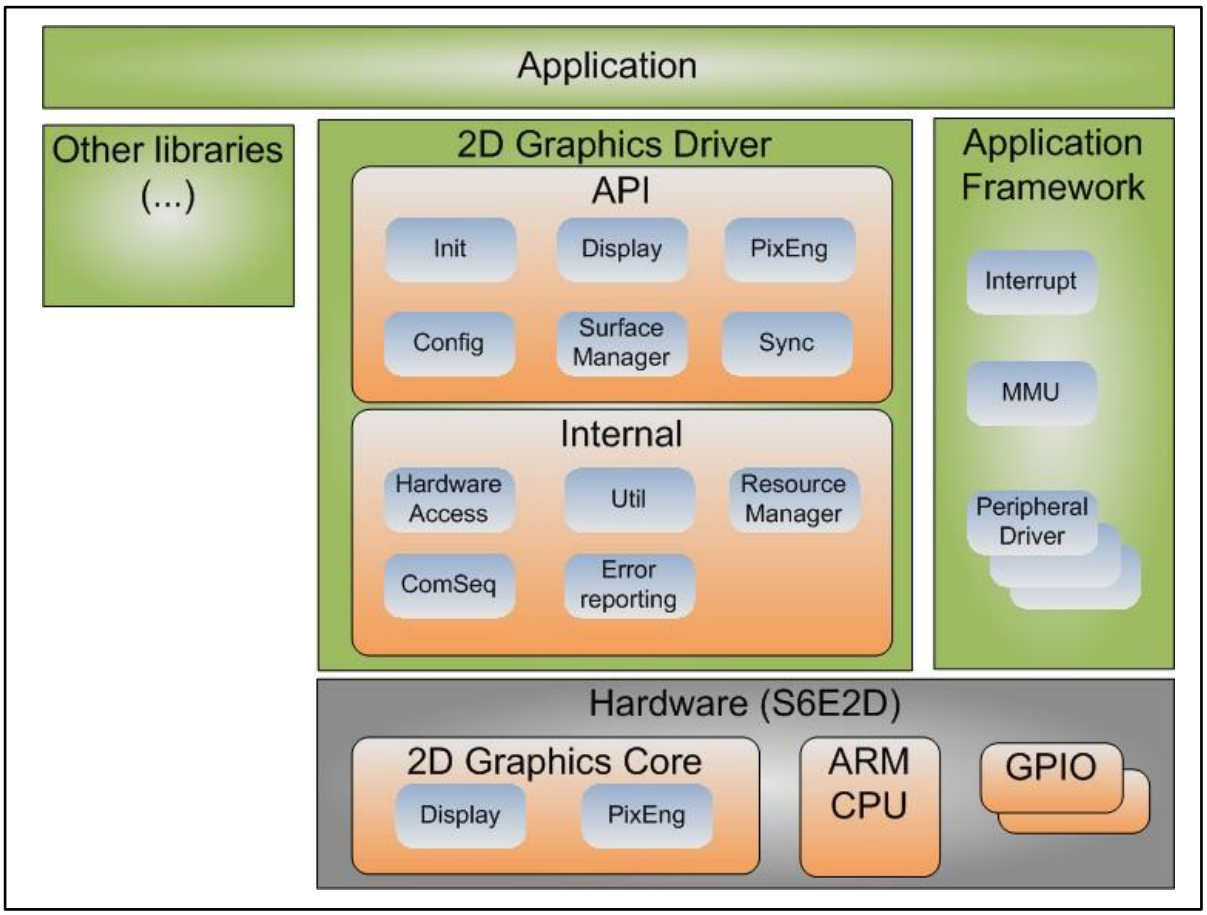
2.1 Target system

The 2D Graphics core and its encircling graphics sub system is a hardware sub-component of an integrated SOC like S6E2D.

Beside the graphical sub-system the chip supports many different peripherals.

The following image shows the basic SOC hardware and software components required to run a typical application.

Figure 2-1 Hardware and software components



2.2 About this document

This document describes the API and usage of the 2D Graphics Driver required to use the 2D Graphics core.

The document does not describe the required application framework or the usage of other peripherals apart from the 2D Graphics core.

2.3 Copyright

3. Getting Started

3.1 Installation

Unpack the archive to a location of your choice.

This package contains all headers, libraries and documentation needed to develop graphics applications for the S6E2D Graphics Hardware. The top level directory contains the following directory structure:

- **00_s6e2dh_demeter_sw_framework** - Cypress FM4 application template, startup code and peripheral drivers.
- **01_bin** - Graphics Core libraries.
- **02_driver** - API header files.
- **04_sample** - Sample application source code.
- **05_util** - Utility library source code used by some of the samples.
- **08_tool** - Tools used by some of the samples.
- **11_doc** - User Documentation.

Building examples: For each sample application there is a subdirectory IAR, ARM or GNU (depending on the supported platform) containing a project file for the respective tool chain (e.g., IAR Embedded Workbench 7.10 or Keil uVision).

3.2 How to run an application

If the toolchain provides flash support for both internal flash and external hyper flash, the tutorial applications can be started from the debugger. Otherwise an appropriate flash programmer is required to download the code and image data to the S6E2D Starter Kit.

3.3 Writing an application

The following steps list the typical flow of an application using the 2D Graphics hardware:

1. Initialize the graphics driver (see Driver Initialization API).
2. Open the display (see Display API).
3. Create one or more windows (or 'layers') for each display (see Display API).
4. Use the Surface API to describe source and target frame buffers (see Surface API).
5. Use any of the APIs described below to create and manipulate graphics content (see Pixel Engine API).
6. Close all created windows (see Display API).
7. Close the display (see Display API).
8. Uninitialize the driver (see Driver Initialization API).

4. Overview

4.1 2D Graphics Driver parts

As shown in the Introduction the 2D Graphics Driver consists of different modules. The following sub-pages will give an overview about the function of some of these modules. Beside these overview pages this document includes a detailed API documentation for each module.

- Surface Overview
- Display Overview
- Overview Pixel Engine (PixEng)
- Synchronization Overview
- Error Reporting Overview

4.2 Other topics

- Memory Management
- Coordinate System Hints
- Image Compression
- Images With Color Index Table

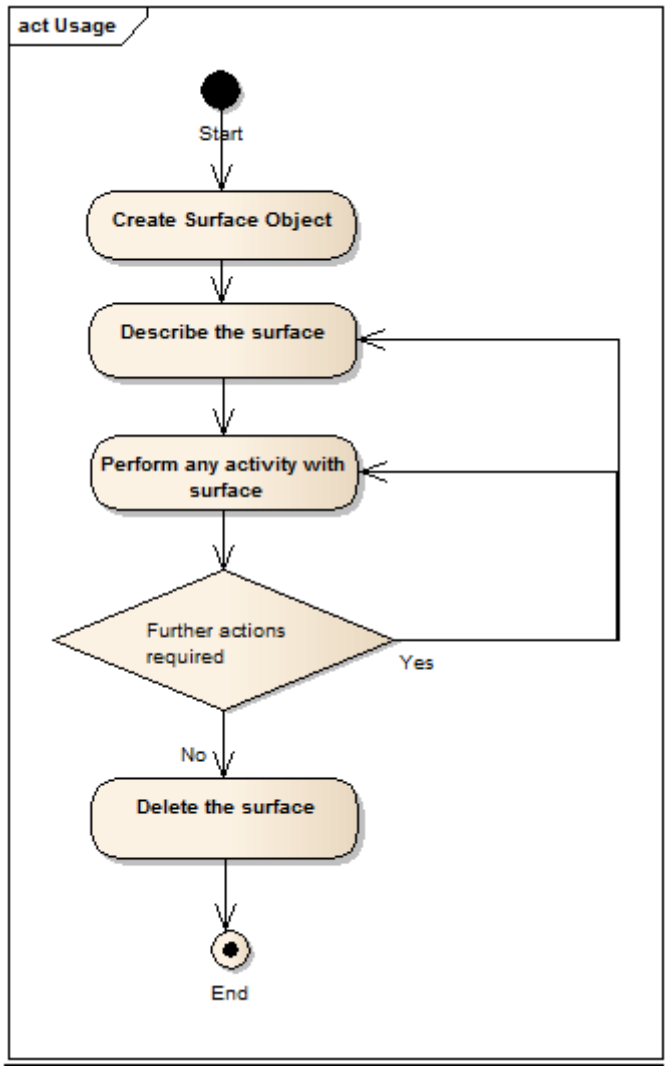
4.3 Surface Overview

4.3.1 Surface objects

The 2D Graphics Driver uses ’surface objects’ to store information about video memory blocks representing an image, a frame buffer and similar things. That means the surface contains the related information about memory address(es), color format, dimension, compression format and more.

The following diagram shows the generic usage of a surface object.

Figure 4-1 Surface usage



Note:

- *Not all hardware blocks can operate with each surface format. Please check the related API description about the supported formats.*

See Surface API for details.

4.4 Display Overview

The 2D Graphics core has one display controller that can be connected to a screen (in the following named Display). The Display has a constant background color "BG Color".

Up to 9 different frame buffers can be used to show content at individual rectangles (called Windows) on the display. The Windows are arranged in a defined z-order which is determined by the layer id and sub-layer id (specified in the properties of each window).

The display controller of the S6E2D device supports up to 2 layers. One of them supports up to 8 sub-layers. It means it is possible to open 8 windows with the same layer id. To use the sub-layers the related window must be opened with the MML_GDC_DISP_FEATURE_MULTI_LAYER feature request. Windows that share the same layerId are called Multi-Window: up to 8 windows with identical layerId, but with different sub_layerIds (also specified in the properties of each window). Multi-Windows that overlap cannot be

blended with each other, they are drawn opaque, (i.e., only the content of the window with the highest sub_layerId is visible).

Windows that overlap can be drawn opaque (only the highest layer is visible) or they can be blended using up to 2 blend units.

Overlapping Windows with the different layer ids can be blended with each other.

Overlapping Windows with the same layer id cannot be blended with each other. Only the content of the window with the highest sub-layer id will be used for the layer blend operation.

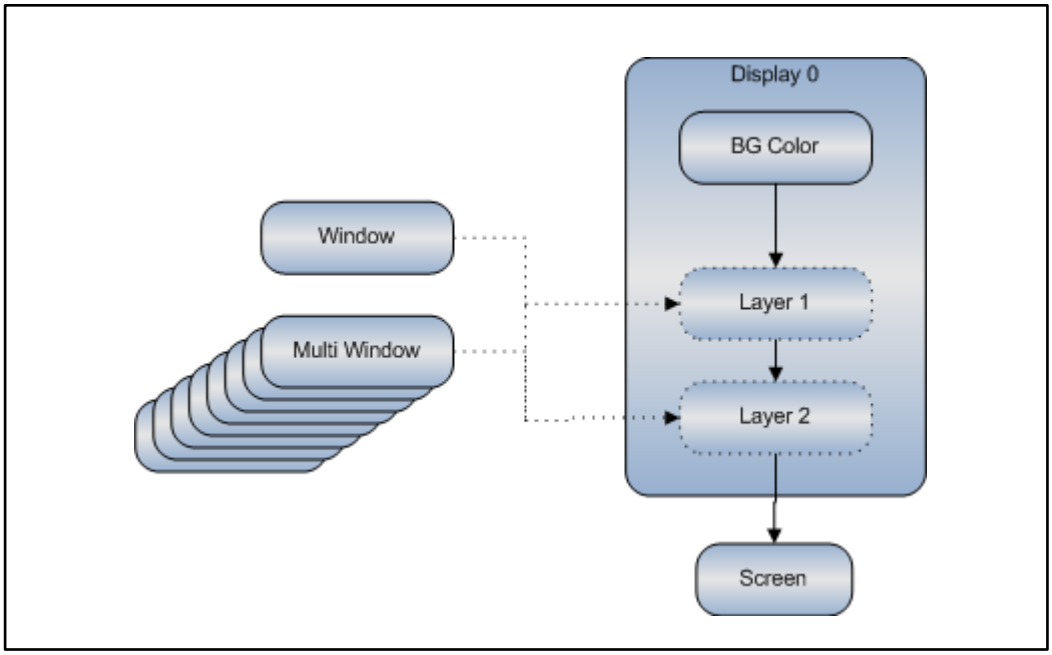
Note:

- *Please note that the hardware manual uses a different wording compared to the software manual and API. The following table describes the different meaning:*

Hardware manual naming	Software manual naming	Description
Background Plane	Background (BG) color of display	Each display controller has the capability to generate a full screen constant color.
Foreground Plane	Layer	See glossary layer
Layer	Sub-Layer	See glossary sub-layer

When a Window is created it is assigned to a Display.

Figure 4-2 S6E2D display unit



This architecture allows creating complex scenes with low VRAM usage and low memory usage. The following example shows a possible scene for one display controller for a device with 5 blend units and up to 26 windows. The S6E2D cannot handle such complex scene however the sample shows the idea behind the layers and windows:

Figure 4-3 Sample display scene

A single layer architecture requires to render all details for each frame. It requires much CPU time and VRAM. The display window concept allows the following architecture visualized as perspective view:

- The gray colored window shows a background layer that might be a compressed 8 bpp indexed image buffer because the content is static.
- The yellow colored windows represent the second layer. Each separate window may be rendered and updated with a different frame rate and color format.
- The green colored windows represent the next layer blend level.
- The red colored windows are the most top windows. In this case they show static 4 bpp indexed images and can be independently switched and faded.

Figure 4-4 Perspective view to the scene

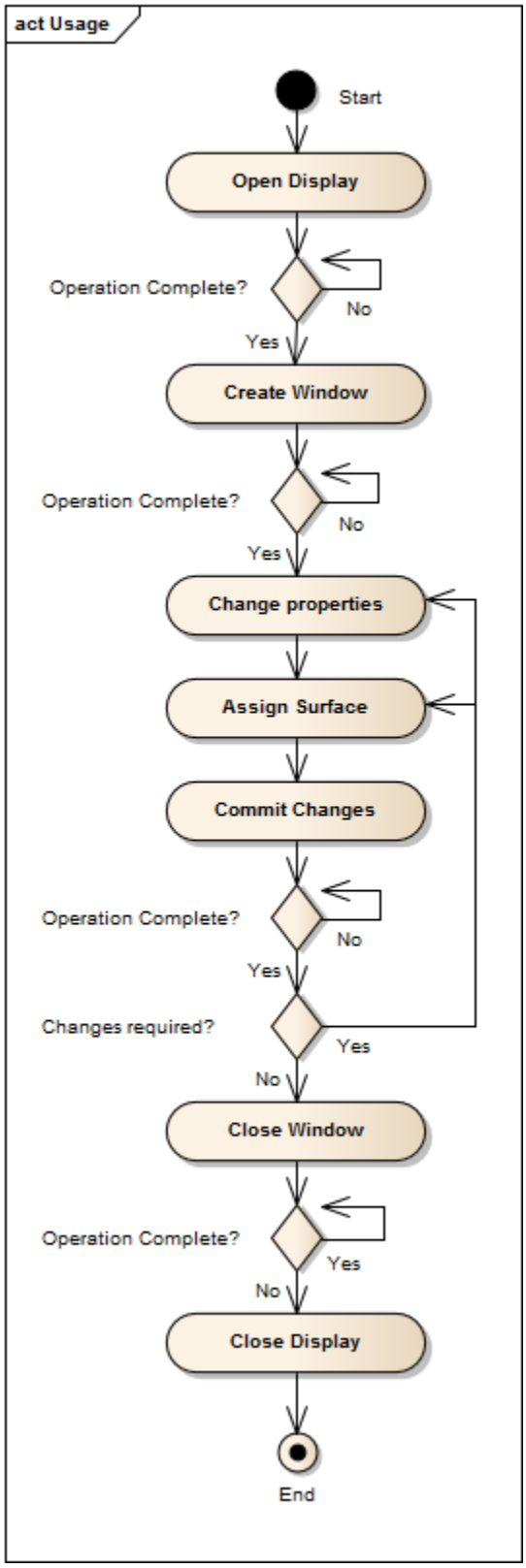
All windows support a minimum functionality: show a image or frame buffer with red, green, blue or gray and optional alpha information. The color and optional alpha information will be read from a continuous video memory block with a defined width, height and stride. The bits per pixel (BPP) can be 1, 2, 4, 8, 16, 18, 24 or 32. The mapping to the color or alpha channels can be selected freely. The images can be used with Simple Transformation.

Some windows support special features beside this standard feature set. The different window names in the image above reflect such features. The usage of such an advanced feature may restrict other Display or Windows properties.

4.4.1 Usage

The following image shows the steps required to use one 2D core Display Controller with one Window.

Figure 4-5 Activity diagram



The Display API lists all supported features and the related restrictions.

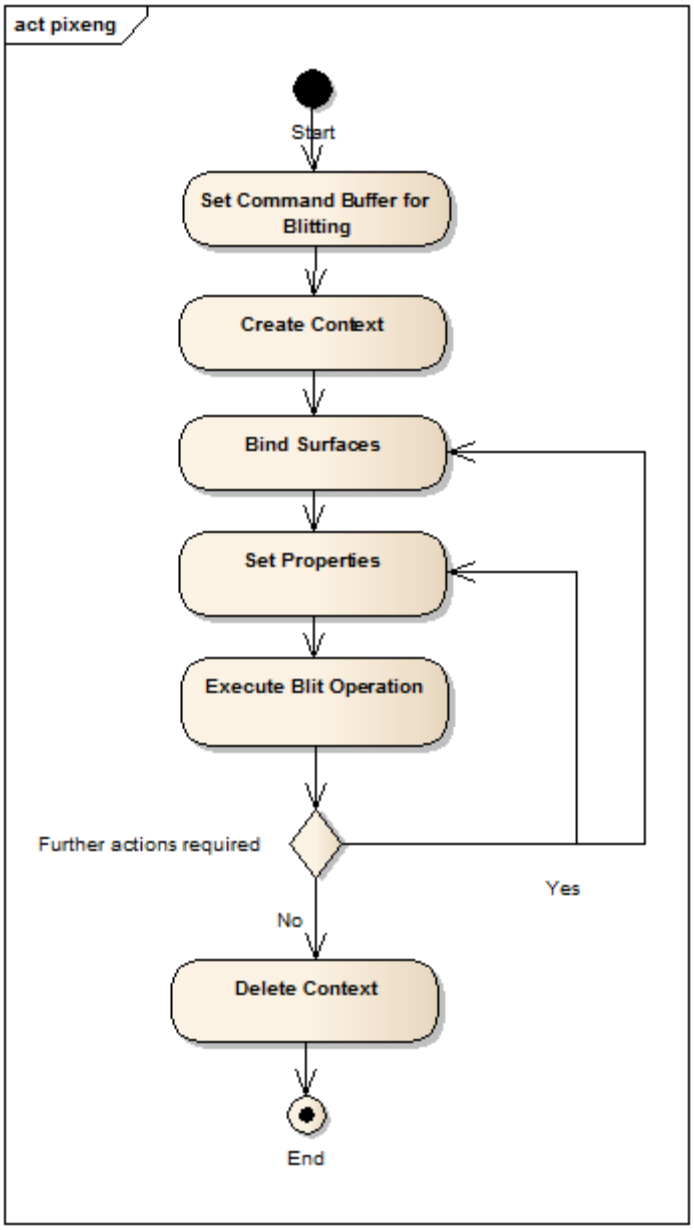
4.5 Overview Pixel Engine (PixEng)

4.5.1 Pixel Engine

The Pixel Engine is a hardware IP that efficiently performs pixel operations on two-dimensional memory blocks. It reads simultaneously from up to three source rectangles, executes pixel processing operations on these and stores the result in another rectangular memory region.

The Pixel Engine functionality is covered by the Pixel Engine API of the 2D Graphics Driver. The Pixel Engine API uses the concept of ’surface objects’ and ’context objects’ to perform all operations. Surface objects are created and bound to a context to perform blit operations to the memory and deleted when no longer needed. A context needs always a surface bound to the STORE target where the resulting pixel data will be stored. Depending on the requested operation a SRC, DST and MASK surface must also be bound to the context. SRC, DST and MASK surfaces define the pixel sources for a blit operation. A surface object can be associated to a memory address to operate with this memory. It is also possible to use a surface without an attached memory address and use it as a blit source. In this case only some properties such as the clear color and geometry are used.

Figure 4-6 PixEng usage



The active API calls (processing and writing pixel data) of the Pixel Engine API are mmlGdcPeFill and mmlGdcPeBlt. The mmlGdcPeFill call with a previously attached store surface can be used to fill a buffer.

mmlGdcPeBlt can be used for all other operations like copying, scaling, rotation, blending and color manipulating processing and combinations of them. The surfaces bound to the context and the properties set to the context define the requested operations. The following table shows the required and optional surfaces to perform an operation.

Operation	Output (required)	Input (required)	Optional with Blend operation	Optional with ROP2 or External Alpha
Fill	STORE	-	-	-
Copy, Scale, Rotate	STORE	SRC	DST	MASK
ROP3	STORE	SRC, DST, MASK	-	-

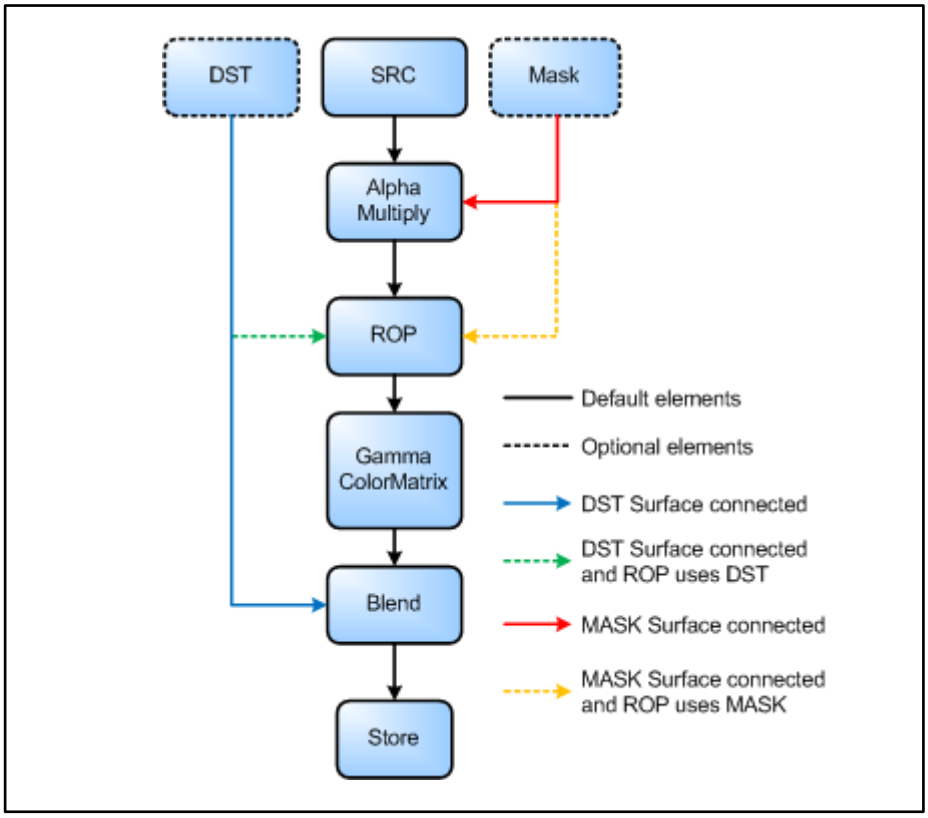
Note:

- *Please note that the hardware manual uses a different wording compared to the software manual and API. The following table describes the different meaning:*

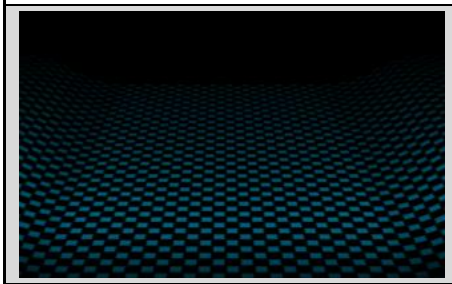
Hardware manual naming	Software manual naming	Description
Pixel Engine	Pixel Engine (PixENG)	Please note that the Pixel Engine defined in hardware manuel has a diffrent meaning. See glossary PixEng.

The processing flow for the related operations are visualized in the following image:

Figure 4-7 Processing flow for blit operations



Example: The following images are used

Surface 1 color channels	Surface 2 color (RGB) channels	Surface 2 alpha (A) channel (A)	Surface 3 alpha (A) channel
			

A blit operation will show the following result depending on the bounded surfaces

Operation	Copy	Blend Operation	Blend with external alpha
Bound surfaces	STORE=Surface 1 SRC=Surface 2	STORE=Surface 1 SRC=Surface 2 DST=Surface 1	STORE=Surface 1 SRC=Surface 2 DST=Surface 1 MASK=Surface 3
Result			

All geometry related settings like translation, scaling or mirroring can be calculated using matrices. Each source surface can get its own matrix, so different translations for source buffer and blend destination are possible. The coordinate center is per default the lower left corner of each surface so a simple copy instruction will copy the source surface to the lower left corner of the store surface.

Furthermore, the connected surfaces (STORE, SRC, DST and MASK) can have different properties like color format dimension or compression. Not all properties are supported for each pipe configuration. Surface properties may also restrict other blit features. For instance it is not possible to rotate a compressed image. The mmlGdcPeBlt call reports an error if the given properties cannot be applied to the hardware. In this case the application developer must simplify the blit operation or may split it into 2 separate blit instructions with a temporary buffer.

The following table gives an overview about supported surface properties and features:

Target	Surface and context properties
STORE	– All RGBA formats.
SRC	– All RGBA formats. – Geometry operations like translation, scaling, rotation and perspective transformation. – Decompression or indexed color. (Restriction: DST must not use these features, only scale and translation operations are supported) – Warping. (No geometry operations possible)
DST	– All RGBA formats. – Geometry operations like translation, mirroring and simple rotation (multiple of 90 degree). – Decompression or indexed color. (Restriction: SRC must not use these features, no mirroring or simple rotation)
MASK	– All RGBA formats except 18 bpp. – Geometry operations like translation, mirroring and multiple of 90 degree rotations – Scaling. (Restriction: SRC must use the same scale factors)

Note:

- *The pixel operations may not be finished after a mmlGdcPeFill or mmlGdcPeBlt call. That means the involved buffers may still be in use. Please use synchronization objects or simply mmlGdcPeFinish to ensure that all operations are complete.*

Pixel Engine operations can be queued by the driver to enhance performance especially in a multi-threading environment. The fast execution especially of long processing commands can be forced by an mmlGdcPeFlush call.

For more details about the usage of the Pixel Engine API see the tutorials and the respective sample code that are part of this driver documentation.

4.6 Synchronization Overview

4.6.1 Processing Units

The S6E2D hardware consists of several independent, parallel running units. The driver is designed so that applications can use this parallel processing also in single threaded environment. The driver distinguishes the following processing pipelines:

- CPU: The ARM core executing the program code.
- The PixEng processing block. All blit instructions for this pipeline will be pushed by the driver into the Command Sequencer queue. That means the application (the CPU) can initiate many fill and blit commands in a series, without having to wait for completion of these commands. The graphics hardware starts operating in parallel, typically it requires more time to process the pixels than to setup the processing by the CPU.
- Windows: All graphics hardware layers are represented as Windows in the driver and handled as separate processing units. Changing properties like the frame buffer address of a window must be committed using mmlGdcDispWinCommit(). The new properties become active with the next frame start on the display. A commit instruction may block the CPU, if the previously called mmlGdcDispWinCommit() is not yet active in the HW. That means if two calls of mmlGdcDispWinCommit() are called one after the other, the second call will be blocked until the next frame start. This behavior can be changed using the MML_GDC_CONFIG_ATTR_DISPLAY_NOBLOCK attribute of mmlGdcConfigSetAttribute() or by using the driver synchronization API.
- Display: The display (more precisely: display controller) is also handled as a separate unit.

4.6.2 Synchronization

The Synchronization API provides mechanisms to synchronize the processing blocks. This is done through sync objects. A sync object describes a sync condition, (e.g., a certain image buffer operation that has to be completed). Sync objects are managed through the Synchronization API, which provides functions to reset sync objects and to wait for a sync condition to become true. Setting a sync condition (in a sync object) is done by the component that owns the sync type. For example, the Display API provides a function to write the sync condition "Surface to be displayed is actually shown on the screen" to a sync object. Waiting for a sync condition can be done by an application (as described above), which is called a "client wait", but also in a graphics processing pipeline without intervention by the application. This is called a "server wait". Server waits are implemented by the component that owns the graphics processing pipeline. For example, the Pixel Engine API provides a function to submit a sync condition to the Pixel Engine command queue (queue to hold the submitted PixEng operations). PixEng operations submitted after the sync, will only be executed after the sync condition becomes true.

Following are a few examples to illustrate the use of sync objects:

- An application renders 2D graphics onto the screen using double-buffering. It can use sync objects to make sure a pixel buffer has already been displayed, (i.e., is free to render a new 2D graphics into it).

The following processing unit events can be used to generate a sync condition:

- Display Controller VSync (new frame started): see mmlGdcDispSyncVSync()
- Window mmlGdcDispWinCommit() is executed: see mmlGdcDispWinSync()
- Previously committed PixEng operations are finished: see mmlGdcPeSync()

The following possibilities for sync server waits exist within the 2D Graphics Driver:

- The Window mmlGdcDispWinCommit() may wait for a sync condition: see mmlGdcDispWinWaitSync()
- The Pixel Engine command queue can wait for a sync condition: see mmlGdcPeWaitSync()

The CPU can check a sync condition:

- Check sync condition: see mmlGdcSyncWait()

4.6.3 Sample use cases

4.6.3.1 Double buffered window

A typical application must render a new frame content for each display loop. Double buffered frame buffers are used to render the next frame in a background buffer while the foreground buffer memory is read by the display controller. The following sample code shows how the application can use the 2D Graphics Driver Synchronization API to realize a double buffered Window:

```
// This structure contains the objects required for a double buffered window.
struct   DOUBLE_BUFFERED_WINDOW
{
    MML_GDC_DISP_WINDOW win; // the window handle
    MML_GDC_SURFACE_CONTAINER sFramebuffer[2]; // Two buffers described as surface objects.
    MML_GDC_SYNC_CONTAINER sync; // A sync object.
    MM_U08 id; // An id storing which buffer is currently the foreground buffer.
};

// This is the draw function for the window including buffer swap and synchronization.
MM_ERROR draw(DOUBLE_BUFFERED_WINDOW *pdbWin)
{
    // Return if the last render operation is still ongoing.
    if (mmlGdcSyncWait(&pdbWin->sync, 0) == MML_ERR_GDC_SYNC_TIMEOUT)
        return MML_OK;

    // Bind new background buffer to render context.
    mmlGdcPeBindSurface(&ctx, MML_GDC_PE_STORE |
    MML_GDC_PE_DST,    &pdbWin->sFramebuffer[pdbWin->id]);
    // Render the next frame
    mmlGdcPe..
    mmlGdcPe..
    mmlGdcPe..
    // Get a sync object for the last blit operation ...
    mmlGdcPeSync(&pdbWin->sync);
    // ...and push it in the windows pipe. (It ensures that the new buffer becomes
    //visible after the last blit is executed in the hardware.)
    mmlGdcDispWinWaitSync(pdbWin->win, &pdbWin->sync));

    // Swap  the  foreground  and  background  layer on  display.
    mmlGdcDispWinSetSurface( pdbWin->win,
    MML_GDC_DISP_BUFF_TARGET_COLOR_BUFF, &pdbWin->sFramebuffer[pdbWin->id] );
    // Commit changes.
    mmlGdcDispWinCommit( pdbWin->win );
```

```
    // Get a sync object for this commit function for the next loop.
    mmlGdcDispWinSync( pdbWin->win, &pdbWin->sync );

    // Switch foreground and background buffer id.
    pdbWin->id = (pdbWin->id == 0) ? 1 : 0;

    return MML_OK;
}

// Here is the calling render loop.
main
{
    DOUBLE_BUFFERED_WINDOW win_struct;

    // Init variables, open window, ...
    // Bind new background buffer to render context.
    mmlGdcPeBindSurface(&ctx, MML_GDC_PE_STORE |
    MML_GDC_PE_DST,          &win_struct.sFramebuffer[win_struct.id]);
    // Render the first frame.
    mmlGdcPe..
    mmlGdcPe..
    mmlGdcPe..
    // Reset the sync.
    mmlGdcSyncReset(&win_struct.sync);
    // Get a sync object for the first blit operation.
    mmlGdcPeSync(&win_struct.sync);

    while()
    {
        // Proceed with any non-graphics related operations.
        do_anything();
        // Call the render routine.
        // Note that the draw function will only render new content if a frame swap
        // was executed. Otherwise the draw function will return immediately so that
        // do_anything() is called again.
        draw(&win_struct);
    }
}
```

The draw() function starts rendering if the previously rendered buffer becomes visible. The application can push all render instructions in the queue, adds a sync instruction that the next buffer swap has to wait for blit complete and assigns the new buffer to the window. Afterwards the CPU can handle other tasks. Please note that the command sequencer queue (see mmlGdcSysSetInstructionBuffer()) must be big enough to store all blit operations.

4.6.3.2 Single buffered window

As double buffering requires more memory, it is worthwhile to consider whether a single buffer is sufficient for a specific application. In this case care must be taken that rendering does not affect the part of the window that is currently read by the display controller to avoid tearing. A simple technique is to do the rendering completely in the blanking period of the display (as demonstrated in the Speedometer sample). A more sophisticated approach splits the frame buffer into several regions and updates only the region that is currently not read by the display controller (as demonstrated in the Chart sample).

See Synchronization API for details.

4.7 Error Reporting Overview

This API provides functions to configure the reporting of ERROR, WARNING and INFO messages. The level of these messages can be specified per module.

See Error Reporting API for details.

4.8 Memory Management

Different to many other graphics drivers the 2D Graphics Driver for S6E2D does not include or use any memory management routines (dynamic memory usage). However memory is required for different functions:

4.8.1 System Memory:

System Memory is a memory block assigned to the CPU as operating memory for OS and application. The driver requires some static memory blocks that should be assigned by the linker to this block. In the reference implementation, S6E2D's SRAM0 is used for this purpose.

2D Graphics hardware blocks can read and write system memory. Typically the 2D Graphics components should not be configured to access system memory because especially frame buffer and similar operations are optimized for the VRAM access.

4.8.2 Video Memory (VRAM):

The Video Memory is a dedicated memory block inside the Graphics hardware designed to store graphical content. The VRAM is also used as command list buffer. Therefore, it is required that the CPU must also have access to the VRAM.

4.8.3 Flash Memory:

Program code and image data are typically read from (embedded or external) flash memory. In most of the example applications, the embedded flash is used for code and external (hyper) flash for data used by the graphics engine ("RES_SECTION"). This is accomplished by a linker directive (see flash_resource.h).

4.8.4 Physical Address - Virtual Address

In this document, in particular in the Surface API description, the terms Virtual Address and Physical Address are used.

For S6E2D devices the physical and virtual address of a register or memory block are identical because the hardware does not contain a Memory Management Unit. Such a Memory Management Unit is typically used by complex operating systems to assign different applications or drivers individual (virtual) memory ranges different from the real physical addresses. If the 2D Graphics Driver is used in such a system an address type differentiation and translation is required and therefore the driver partly supports both types.

Because the 2D Graphics Driver was developed using a software model of the 2D core that requires a differentiation of physical and virtual address, some tutorial examples use an address translation macro. The macro does not change the address for the final 2D Graphics Driver.

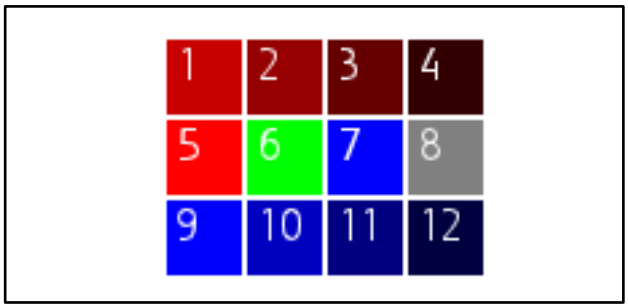
4.9 Coordinate System Hints

Driver APIs for graphical operation use different coordinate systems. The following definitions are used inside the 2D Graphics Driver:

4.9.1 Surface (Image) buffer

Images (described by a surface) are always line based from top to down, left to right. That means the first bits in a surface memory buffer describe the upper left pixel, the next bits describe the pixel right of the first, and so on.

Figure 4-8 Zoomed image with pixel enumeration



4.9.2 Display coordinates

Analog to images the display coordinate system always starts at the top left pixel.

4.9.3 PixEng coordinates

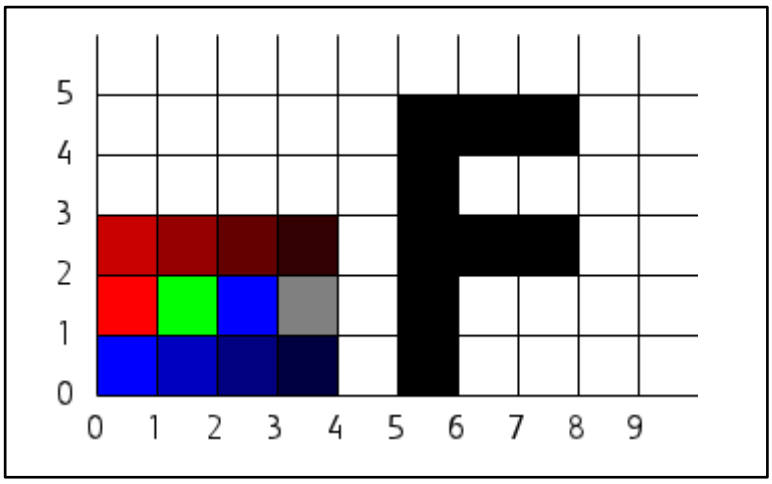
For compatibility reasons the PixEng coordinate system starts per default with the bottom left pixel.

Note:

- *The very first pixel starts at 0.0, 0.0 and ends at 1.0, 1.0. That means the geometrical center of this pixel is 0.5, 0.5.*

The following image is the result of a copy instruction of the 4 ∗ 3 pixel image above with offset 0, 0.

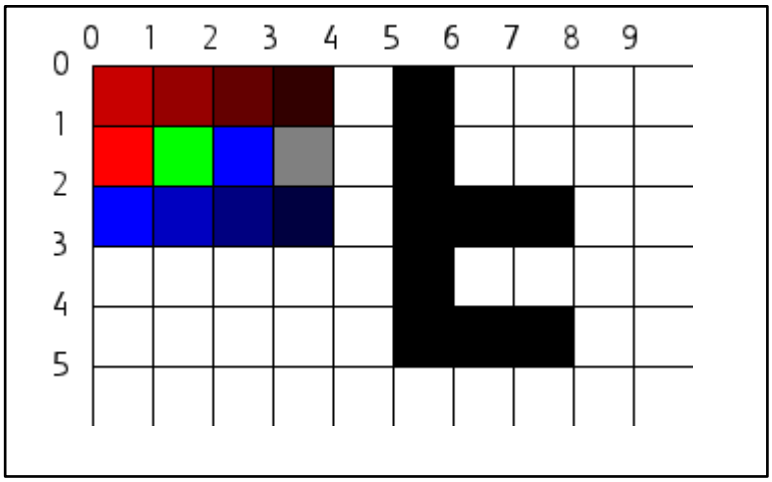
Figure 4-9 Zoomed blit and draw result with bottom left coordinate system setting



For some use cases it is much simpler to use the same coordinate system like the display. Besides this some graphics formats (e.g., SVG) and APIs use the opposite coordinate system. That's why the 2D Graphics Driver supports the mirrored coordinate system too and the user can switch the coordinate handling to top left zero point.

The same image rendered above will now show the following result:

Figure 4-10 Zoomed blit and draw result with top left coordinate system setting



4.9.4 Matrix helper functions

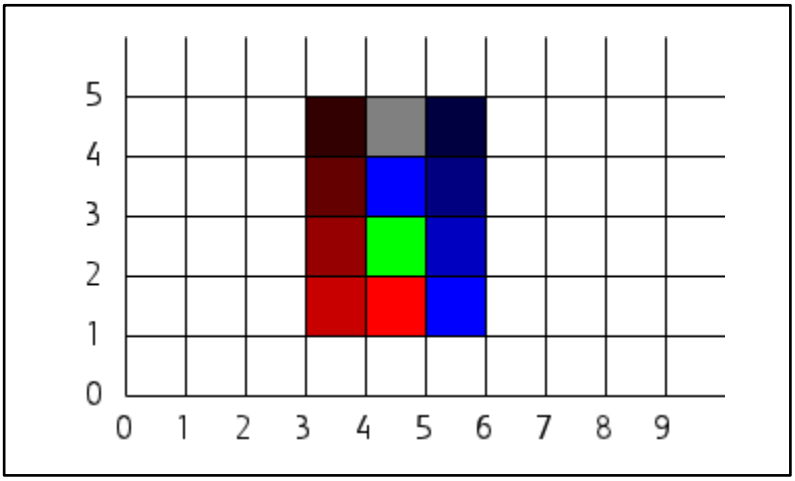
The 2D Graphics Driver comes with many tutorial samples and sample code. For geometrical operation the utility part includes matrix calculation helpers. Different matrix formats are available

- Mat3x2: This matrix format can be used for 2 dimensional operations like translation, scaling and rotation.
- Mat3x3: This matrix format must be used for the API in the blit path for the source image if a "3D" operation is required.
- Mat4x4: The 4x4 matrix is just a helper format. The related functions are basically similar to other "3D" render APIs like OpenGL. However, the depth information is not used, so the 2D Graphics Driver API does not support this matrix format. An application can anyway use these helper functions for the view calculation, because the matrix result can be converted into a 3x3 matrix by removing the depth (z) parts from the matrix.

The following example shows the required Mat3X2 operations to rotate the image above at the center of second pixel in the second line and blend the result to a target. The rotation center of the source pixel will be located at the center of pixel 4, 2 in the target.

```
//reset the matrix
utMat3x2LoadIdentity(mat);
//translate to target coordinates
utMat3x2Translate(mat, 4.5f, 2.5f);
//90 degrees rotation
utMat3x2Rot(mat, 90.0f);
//translate to center of pixel 1, 1 in source coordinate system
utMat3x2Translate(mat, -1.5f, -1.5f);
```

Figure 4-11 Zoomed blit result with matrix operation (bottom left coordinate center)



4.10 Image Compression

To reduce the amount of required memory the 2D core HW supports compressed images.

4.10.1 Compression Formats

The following compression formats for pixel buffer are supported by the 2D core.

Name	Features	Limitations	Recommended use case
Run-length coded MML_GDC_SURF_COMP_RLC	– Lossless compression. – Backward compatible to legacy devices. – RLC compression can be used in combination with indexed color.	– Only supported as read buffer for blit operations and as window content. – Rotation or mirroring is not supported.	– Compression of source images with long line parts with constant color.

Name	Features	Limitations	Recommended use case
Run-Length Adaptive MML_GDC_SURF_COMP_RLA	– Lossless compression. – Good compression results for images with smooth content borders.	– Only supported as read buffer for blit operations and as window content. – Rotation or mirroring is not supported. – A compressed buffer must not exceed the window dimension.	– Compression of source images.
Run-Length Adaptive Dithering MML_GDC_SURF_COMP_RLAD	– The 2D core HW can read and write this format. – Maximum buffer size can be calculated (1).	– Lossy compression. – Rotation or mirroring is not supported. – A compressed buffer must not exceed the window dimension.	– Compression of source images with size limitation.

4.10.1.1 (1) Calculation of required buffer size for RLAD compression

The following formula can be used to calculate the maximal required buffer size:

pixel_size = cbpc0_max + cbpc1_max + cbpc2_max + cbpc3_max

header_size = (cbpc0_width + cbpc1_width + cbpc2_width + cbpc3_width) + (bpc0 + bpc1 + bpc2 + bpc3)
num_header = ceil(frame_width / 8) * frame_height

buf_size = num_header * header_size + frame_width * frame_height * pixel_size

buf_words = ceil(buf_size / 32)

- bpc0/1/2/3: ComponentBitsRed/Green/Blue/Alpha (see MML_GDC_SURF_ATTR_COLORBITS).
- cbpc0/1/2/3_max = RLADCompBitsRed/Green/Blue/Alpha (see MML_GDC_SURF_ATTR_RLAD_MAXCOLORBITS).
- cbpc0/1/2/3_width = floor(log2(bpc0/1/2/3)) +1 or 0 if the component size is 0.
- frame_width/height = dimension of input frame.

Some typical setups and resulting compression rates (compressed/uncompressed) for RGB888 image data: RLADCompBitsRed/Green/Blue

- 4/5/4 => 73 %
- 3/4/3 => 61 %
- 2/3/2 => 48 %

Note:

- *Images compressed with MML_GDC_SURF_COMP_RLAD may result in a smaller size however also for worst case images the maximum size will not be exceeded.*

4.10.1.2 How to create compressed images

The Tutorial Utility Library contains a Utilities for the compression part providing sample code to create run-length-coded and run-length-adaptive compressed buffers. Furthermore the 2D Core Driver package contains a windows command line tool "ResourceGenerator.exe" that can be used to convert a png image into a compressed buffer and store it in a c compliant header file. Afterwards this content can be used by the utSurfLoadBitmap() function to fill a MML_GDC_SURFACE object that can be used for blit or display API functions.

4.11 Images With Color Index Table

To reduce the amount of required memory the HW supports images with indexed colors. In this case the image requires 2 buffers:

- One buffer contains all possible colors for this image: the color table or "color look up table".
- The second buffer is the typical image buffer. But for each pixel in the image, it only stores an index pointing to a color in the color table.

4.11.1 Alpha support

Index images can also include per pixel alpha values to control the transparency of the addressed color. The alpha information can be stored either in the image buffer beside the index pointer or it can be part of the color table.

4.11.2 Image buffer

Like other image buffers also image buffer for indexed images can use different sizes. Depending on the bit width of the index pointer the image can store a defined maximum of different colors. Beside the index pointer the image buffer may also contain alpha bits. The sum of alpha and index bits must be 1, 2, 4, 16, 24 or 32. The index bits must start at bit position 0.

The following table shows some possible pixel buffer color formats for indexed images. Only the size of red channel in a surface defines the index width. The green and blue channel definition is not used for such images. Therefore a short format RGB8 is equivalent to 8 bit index.

Short format	Bit per pixel	Index bits	Alpha bits	Maximum of visible colors	Use case
RGB8	8	8	0	256	Images without per pixel alpha and a maximum of 256 different colors.
RGB4	4	4	0	16	Images with 16 colors only (please note: the palette may include an alpha bit too. That's why it is a possible use case to address 15 visible colors and 1 transparent color).
A8RGB8	16	8	8	256	Images with per pixel alpha and a maximum of 256 different colors.
A3RGB5	8	5	3	32	Images with 8 levels of transparency (alpha) and a maximum of 32 different colors.
RGB1	1	1	0	2	All images with only 2 different colors.

4.11.3 Color table

The color table can store up to 256 different colors. Each entry defines the RGB and optionally the alpha value. The maximum number of bits to store these values are 24 bit per entry. Therefore supported color table formats are R8G8B8 or R6G6B6A6 but R8G8B8A8 with 32 bit per color is not supported.

4.11.4 Surface properties for indexed images

Like other images also indexed images are described by surface objects. In this case the application must define the format and address of the image buffer and color table buffer.

4.11.5 Index images for blit operations

Blit operations like blending a traffic sign to a target buffer can be proceeded like operations with standard RGB(A) images because the surface contains the required information.

Note:
The following restrictions exist for indexed images:

- *Surfaces describing an indexed image cannot be used as STORE buffer.*
- *It is not possible to use indexed images as DST and SOURCE surface for one blit.*
- *Indexed images cannot be scaled or rotated.*

4.11.6 Index images for the windows

The display hardware can directly show indexed images. In this case the application needs to request the MML_GDC_DISP_FEATURE_INDEX_COLOR feature while opening the window.

However, there are some restrictions if the window also uses the MML_GDC_DISP_FEATURE_MULTI_LAYER feature:

- All windows with the same layer ID must use the same index width (red channel bit width) if they show indexed images.
- One layer with up to 8 sub-layer-windows can only store 256 palette entries. So if the index width is 8 for this layer all windows will use the same palette.
- If the index width is smaller than 8 than the palette is split in 2 or more parts and the sub-layer-windows can partly use different palettes. The following rule is implemented in hardware: the upper bits of the 8-bit look-up index are then filled up with the upper bits of the sub-layer index. Example: when a 6-bit color index value is used (= 64 colors), 4 palettes can be stored, each shared by 2 layers (layer 0 and 1 use palette entries 0..63, layers 2 and 3 use 64..127 and so on).

Note:

- *The driver does not check this rule. If the application binds surfaces with different palettes to windows sharing the same hardware palette a wrong image will be the result.*

Like other settings also palettes are shadowed. It means you can commit the binding of a new indexed image surface with a new palette while an old one is still visible. If sub-layer-windows share the same palette it is recommended to hide all windows before the new palette becomes active.

Notes:
Unfortunately the hardware shadow handling for palettes is in some cases not as expected: Each update request in a window group with the same palette triggers the palette swap. For instance

- *2 (or more) sub-windows address the same palette part (e.g., if index width is 8 all sub-windows use the same palette).*
- *Only one window uses the palette.*
- *The commits for all windows without palette may also trigger the palette swap. (for example with index width 8 it means that each commit of a window with the same layer id can trigger the swap and the window with the indexed surface will sometimes show the correct and sometimes wrong colors.)*

To avoid this problem the application must commit a surface with a new palette twice. The two times commits ensure that the shadow palette is filled with the correct color table. In practice the application may call 2 times mmlGdcDispWinCommit() directly. In this case the CPU will be blocked until the first commit is taken over by the hardware (when a new display frame is started). If blocking is not acceptable the application must take care that the next render loop calls the mmlGdcDispWinCommit() for this window. As soon as the application changes properties for this window (e.g., change global transparency for fading) the mmlGdcDispWinCommit() call is anyway necessary.

5. Glossary

Name	Description
Basic Graphics	Term to summarize the functions of the Graphics driver that use the 2D Graphics hardware. Used to differentiate it from other graphics functions like OGL, if supported by the hardware.
Blend Operation	Blend Operation stands for a calculation of output pixels depending on the color and alpha information of 2 input pixels.
bpp	bpp stands for "bit per pixel" and describes how many bits are required to define the color and alpha values on one pixel in an image. Most modules inside the 2D core support the following bpp sizes: 1, 2, 4, 8, 16, 18, 24, 32.
Display	The term display is used to describe the output device showing the content generated by the 2D core. Depending on connected hardware it can be an (LCD) panel, monitor, beamer or similar. In context with the Basics Graphics Driver it is also the short name for the 2D core Display Controller, a part of the 2D Graphics hardware.
External Alpha	External alpha stands for an image buffer containing an alpha channel that is used as transparency information while blending a different color buffer over a background image. The external alpha value will be multiplied with the alpha channel of the color buffer in this case.
Frame buffer	A frame buffer is an image buffer that is typical first used as render target and afterwards it’s content will be shown on a connected display.
Indexed image	An indexed image does not store for each color a separate RGB(A) value but a single index value. This index value points to a color look up table with up to 256 RGB(A) values that will be used as pixel color. Such images can not contain more than 256 different colors but the size of such images are smaller. The alpha channel can be part of the look up table, it can be a separate channel beside the index channel or the image does not contain alpha information.
Layer	A color plane in front of the display background color from the visitors point of view. If multiple planes or layers are supported by hardware they will have a defined z-order. Each upper level can modify the pixel color of the lower level. In the end one pixel on the display can be the blend result of background color and all layer levels.
Physical Address	The Physical Address stands for an address representing the "real" hardware address of a register or start of a memory block or similar. In contrast to Virtual Address this address type is used by 2D Graphics hardware components. See also Physical Address - Virtual Address
PixEng	Stands for Pixel Engine: Part of a chip based on the 2D core components that is responsible for pixel buffer based transformations like copying, rotation, bending and much more.

Name	Description
Pre-multiply	If images contain an alpha channel this alpha channel will be often used for per pixel blending. The required blending formula depends on the way how the color channels RGB are stored in such an image: – Non-pre-multiplied: they contain the original pixel color independent of the alpha value for this pixel. – Pre-multiplied: the stored pixel color is already multiplied with the alpha value of the same pixel. PNG images are often stored as non-pre-multiplied images. An 2D core render buffer is typical a pre-multiplied image. Alpha channel of an image:  Color channels if it is an "non-pre-multiplied" image. Required blend formula: C = Csrc∗Asrc + Cdst∗(1-Asrc).  Color channels if it is a "pre-multiplied" image. Required blend formula: C = Csrc + Cdst∗(1-Asrc).  Expected blend result: 
RLA	Abbreviation for **R**un-**L**ength **A**daptive. This is a lossless compression type of an image supported by the 2D Graphics hardware.
RLAD	Abbreviation for **R**un-**L**ength **A**daptive **D**ithering. This is a lossy compression type of an image supported by the 2D Graphics hardware.
RLAD_Uniform	Abbreviation for **R**un-**L**ength **A**daptive **D**ithering with **Uniform** package size. This is a lossy compression type of an image supported by the 2D Graphics hardware.
RLC	Abbreviation for **R**un-**L**ength **C**oded. Synonym for RLE.
RLE	Abbreviation for **R**un-**L**ength **E**ncoded. This is a lossless compression type of an image supported by the 2D Graphics hardware.
ROP2	Raster Operation with 2 sources. A bit field defines how the color data bits of the first source are be combined with the color data bits of the second source to achieve the final color.

Name	Description
ROP3	Raster Operation with 3 sources. A bit field defines how the color data bits of the 3 source are combined to achieve the final color.
Simple Transformation	We talk about simple transformation if an image source is translated, mirrored and/or rotated by a multiple of 90°. Nearly all 2D core components can do simple transformation while reading an image. For instance a display controller can directly show horizontal mirrored images. ***Note:*** *90°and 270°rotations result in a higher memory read rate. Especially for high resolution displays it is not recommended to use this feature. Compressed images cannot be used with simple transformations.*
Stride	The amount of bytes that must be skipped over to get from one pixel in an image to the pixel with the same horizontal position in the next line of this image.
Sub-layer	A sub-layer is analog to layers an image that is blended over a background in the display controller. A sub-layer is always part of a layer. Different to layers it is not possible to blend overlapping sub-layers together if they share the same layer id. Only the top most sub-layer image pixel will be read from memory and this color information will be used for the layer blend operation.
Surface	Surface stands for an memory object describing one or more memory blocks describing an image. Many 2D Graphics Driver API calls use surface objects for the functions. See also Surface Overview.
Virtual Address	The Virtual Address stands for an address representing the CPU view of an hardware address like a register or start of a memory block. In contrast to the Physical Address this address type cannot be used by 2D Graphics hardware components. See also Physical Address - Virtual Address.
VRAM	Abbreviation for **V**ideo **R**andom **A**ccess **M**emory. This is a dedicated memory with short read and write access time that is designed to store and buffer images. The VRAM can be part of the 2D Graphics hardware block but it is also possible to use external memory as VRAM.
Window	The term Window is used to describe a software object that keeps all parameters to push a rectangular image to the display.

6. Tutorial

6.1 About the Tutorial

The tutorial is comprised of chapters which demonstrate the use and possibilities of the driver API. The tutorial chapters are of different complexity levels, starting with basic chapters to become familiar with the way to use the API. The complexity then progressively increases to provide examples which demonstrate special features and ways to achieve effects with the 2D Graphics hardware.

6.2 Application framework

All sample applications are constructed according to a common scheme, based on Cypress's FM4 application template. Basic setup of peripherals, timers, etc. is handled in main.c, which looks similar for all examples. The following application specific functions are called from main():

- InitIrisExample(): Application specific graphics initialization.
- IrisExampleDraw(): Graphics code executed in a loop, (e.g., once per frame).
- IrisExampleCleanup(): Reset graphics system; interactive applications use the buttons of the FM4 Starter Kit to control the software. This is accomplished by state variables set in ButtonCallback() and passed on to IrisExampleDraw().

6.3 Restrictions

Please note that the sample code for this release may differ from the final version.

Especially the usage of synchronization instructions might not always represent the final version.

6.4 Tutorial chapters

The Tutorial 1: Surfaces_Blit_Display Basic show the basic steps to use the 2D Graphics Driver. It starts explaining surface objects, executes some simple graphical operations to fill a pixel buffer and it ends up showing the rendered buffer on a screen connected to the S6E2D hardware.

The Tutorial: Display Basic demonstrates the capabilities of the display path based on an example showing a navigation solution including different display layers.

The Tutorial: Display_Extended demonstrates buffer swapping technique for multiple windows.

The Tutorial: Speedometer demonstrates an application for creating a speed gauge including a rotating needle. Application uses techniques such as active area and background restoration.

The Tutorial: Chart shows a single render buffer solution.

The Tutorial: Cover Flow demonstrates several PixelEngine features in form of a cover flow.

The Tutorial: Digital Picture Frame demonstrates several PixelEngine features in form of digital picture frame.

The Tutorial: Simple Drawing shows how complex features can be realized in software by combining simple hardware features.

The Tutorial Utility Library collects some functions used in different tutorials.

6.5 Tutorial 1: Surfaces_Blit_Display Basic

6.5.1 Description

This is a very simple start-up application to show the usage of MML_GDC_SURFACE and MML_GDC_PE_CONTEXT.

A surface is required to perform operations in the blit and display path.

Please note: Although all APIs use a MML_GDC_SURFACE parameter you need to declare MML_GDC_SURFACE_CONTAINER objects and use the pointer to these objects for all APIs. It holds the following information:

- Buffer dimension
- Memory address of the pixel data
- Color format

and optionally:

- Compression format and parameter
- Color look-up table parameters

6.5.2 Chapters

1. MML_GDC_SURFACE
2. Initialization
3. Fill with constant color
4. A simple black-and-white image
5. A simple auto-generated pattern
6. Blending two surfaces
7. Bring it to the display

6.5.3 MML_GDC_SURFACE

6.5.3.1 Color format

The color format defines the color depth (bits per pixel) for each color channel (red, green, blue, alpha). A lot of common color formats are pre-defined, for example

- MML_GDC_SURF_FORMAT_R8G8B8A8 (32 bits per pixel, 8 bits for each channel)
- MML_GDC_SURF_FORMAT_R5G6B5 (16 bits per pixel, 5 bits for red, 6 bits for green, 5 bits for blue)
- MML_GDC_SURF_FORMAT_RGB8 (8 bit per pixel for red, green and blue, which means 256 gray scale values)
- MML_GDC_SURF_FORMAT_RGB1 (1 bit per pixel for red, green and blue, which means black-and-white)
- ...

6.5.3.2 Compression format and parameter

The 2D Graphics Core can use different kinds of pixel buffer compression: RLC, RLA, RLAD or RLAD_UNIFORM. The surface holds information about compression format and related properties.

6.5.3.3 Color look-up table parameters

A color lookup table can be assigned to a surface by using the function mmlGdcSmAssignClut. Lookup tables can be used for indexed images: the value for red in the image defines an index of the color lookup table. The color of the table entry defines the pixel color at this point.

6.5.4 Initialization

Driver initialization and 1kB for the command sequencer FIFO.

```
// driver initialization
UTIL_SUCCESS(ret,     mmlGdcSysInitializeDriver(0));

// get virtual addresses (not required for devices without MMU) */
MM_GDC_PHYS_TO_VIRT((MM_ADDR)instructionBufferAddr,    &vInstrBufferAddr);

// recalculate    addresses    using    the    vImgAddr    as    start
MM_GDC_PHYS_TO_VIRT((MM_ADDR)imgAddr, &vImgAddr);
patternAddr    =    (MM_U32)vImgAddr    +    imgSize;
textAddr = patternAddr + patternSize; storeAddr = textAddr + textSize;

// set up an instruction buffer for the command sequencer
UTIL_SUCCESS(ret, mmlGdcSysSetInstructionBuffer( vInstrBufferAddr, instructionBufferSize));
```

6.5.5 Fill with constant color

First of all the store surface surfStore has to be initialized with mmlGdcSmResetSurfaceObject. We use MML_GDC_SURF_FORMAT_R5G6B5 as the color format, which means

- 5 bits for red channel.
- 6 bits for green channel.
- 5 bits for blue channel.

We use 0/0/255/255 (pure blue, non-transparent) as the constant color for the store surface. To setup the blit path, the context has to be reset and the store surface is bound to MML_GDC_PE_STORE.

mmlGdcPeFill finally fills the store surface with the given constant color.

```
// the store surface
mmlGdcSmResetSurfaceObject(surfStore);
// use format 5/6/5
UTIL_SUCCESS(ret,    mmlGdcSmAssignBuffer(surfStore,    storeWidth, storeHeight,
MML_GDC_SURF_FORMAT_R5G6B5, (void *)storeAddr, 0));

// the context must be reset like the surface
mmlGdcPeResetContext(ctx);
// we use a coordinate system starting in the upper left corner
mmlGdcPeAttribute(ctx, MML_GDC_PE_ATTR_ZERO_POINT,
MML_GDC_PE_ATTR_ZERO_TOP_LEFT);
// Bind the surface to the context
UTIL_SUCCESS(ret,     mmlGdcPeBindSurface(ctx, MML_GDC_PE_STORE,    surfStore));

// define the constant color
UTIL_SUCCESS(ret, mmlGdcPeColor(ctx, 0, 0, 255, 255));
```

```
UTIL_SUCCESS(ret, mmlGdcPeFill(ctx, 0, 0, storeWidth, storeHeight));
```

Figure 6-1 Constant color

6.5.6 A simple black-and-white image

Now a second surface surfSrc is needed for the source path. It has to be initialized with mmlGdcSmResetSurfaceObject and filled with the image data; a simple 32 x 32 pixel black-and-white image. As it is a 1 bpp image, MML_GDC_SURF_FORMAT_RGB1 is used for the color format.

The store surface keeps the same, but the new source surface has to be added to the context. mmlGdcPeBlt copies the image to the store surface at position 20/20.

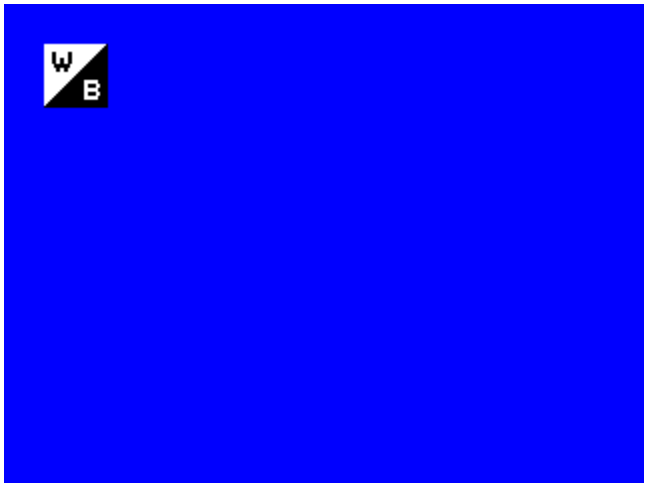
```
// initialization of surfSrc
mmlGdcSmResetSurfaceObject(surfSrc);

CopyToVram(black_and_white, (MM_U32)vImgAddr, imgSize);

// use the black-and-white format
UTIL_SUCCESS(ret, mmlGdcSmAssignBuffer(surfSrc, imgWidth, imgHeight,
MML_GDC_SURF_FORMAT_RGB1, vImgAddr, 0));
// add it to the context as 'source'
UTIL_SUCCESS(ret, mmlGdcPeBindSurface(ctx, MML_GDC_PE_SRC, surfSrc));

UTIL_SUCCESS(ret, mmlGdcPeBlt(ctx, 20.0f, 20.0f));
```

CopyToVram is a small helper function to copy the image data to its destination in the VRAM.

```
static void CopyToVram(const void* data, MM_U32 addr, MM_U32 size)
{
    void *vaddr = (void *)addr;
    MM_GDC_LOCK(vaddr, size, MA_WRITE);
```

```
    memcpy(vaddr,  data,  size);
    MM_GDC_UNLOCK(vaddr);
}
```

Figure 6-2 Black&White image on constant color



6.5.7 A simple auto-generated pattern

Fill the source surface with a simple pattern and copy it to the store surface.

We reuse surfSrc, but we use MML_GDC_SURF_FORMAT_R8G8B8A8 for the surface color format because it is much easier to write the data when each pixel is 4-byte aligned. mmlGdcPeBlt copies the pattern to the store surface at position 35/45, so again the existing content of the store surface is overwritten in that area.

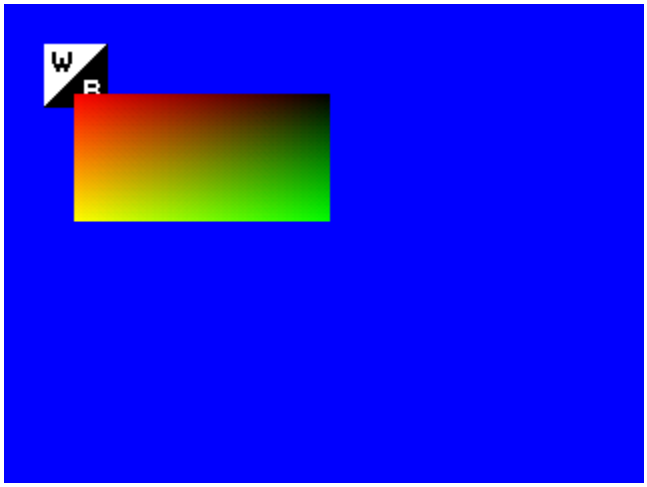
```
// re-use the source surface for the pattern
mmlGdcSmResetSurfaceObject(surfSrc);
UTIL_SUCCESS(ret,  mmlGdcSmAssignBuffer(surfSrc,  patternWidth, patternHeight,
MML_GDC_SURF_FORMAT_R8G8B8A8, (void *)patternAddr, 0));

// create the pattern directly to VRAM
CreatePattern((MM_U32)patternAddr, patternSize, patternWidth, patternHeight);
// add it to the context as 'source'
UTIL_SUCCESS(ret,  mmlGdcPeBindSurface(ctx, MML_GDC_PE_SRC,  surfSrc));

UTIL_SUCCESS(ret, mmlGdcPeBlt(ctx, 35.0f, 45.0f));
```

Creating the pattern:

```
static void CreatePattern(MM_U32 addr, MM_U32 size, MM_U32 width, MM_U32 height)
{
    MM_U32 x;
```

```
    MM_U32 y;
    MM_U32 red;
    MM_U32 green;
    MM_U32 blue;
    MM_U32 alpha;

    MM_GDC_LOCK(addr, size, MA_WRITE);

    for (x = 0; x < width; x++)
    {
        for (y = 0; y < height; y++)
        {
            red = 255 - (2 * x);
            green = y * 4;
            blue = 0;
            alpha   =   255;
            *((MM_U32 *)(addr + (4*((y * width) + x)))) = ((red << 24) | (green << 16) | (blue << 8) | alpha);
        }
    }
    MM_GDC_UNLOCK(addr);
}
```

Figure 6-3 Pattern on Black&White image on constant color



6.5.8 Blending two surfaces

To blend an image with an alpha channel onto the existing store surface, we have to connect surfStore both as destination (input) and store (output).

The second input surface is again surfSrc. It has to be reset because now it holds another image with just 8 bit alpha values. Therefore MML_GDC_SURF_FORMAT_A8 has to be used. We define 255/0/0/255 as the constant color to see the text in red. The alpha channel in the constant color definition has no effect, because it is defined by the image!

```
// re-use the source surface for the text
mmlGdcSmResetSurfaceObject(surfSrc);
UTIL_SUCCESS(ret, mmlGdcSmAssignBuffer(surfSrc, textWidth, textHeight,
MML_GDC_SURF_FORMAT_A8, (void *)textAddr, 0));
UTIL_SUCCESS(ret, mmlGdcPeSurfColor(ctx, MML_GDC_PE_SRC, 255, 0, 0, 255));

UTIL_SUCCESS(ret,     mmlGdcPeBindSurface(ctx, MML_GDC_PE_STORE | MML_GDC_PE_DST,
surfStore));
UTIL_SUCCESS(ret, mmlGdcPeBlt(ctx, 50.0f, 70.0f));
```

Figure 6-4 Blended text

6.5.9 Bring it to the display

To see the surface on the display, we need a to create a display object by calling mmlGdcDispOpenDisplay. Beside this a window is required using mmlGdcDispWinCreate. Finally our surfStore must be set to the window using mmlGdcDispWinSetSurface and mmlGdcDispWinCommit activates the changes.

```
// set up the display
// complete the display params
dispParams.xResolution   =   480;
dispParams.yResolution   =   272;
UTIL_SUCCESS(ret, mmlGdcDispOpenDisplay(&dispParams, &display));

// create a display window and connect the store surface to it
```

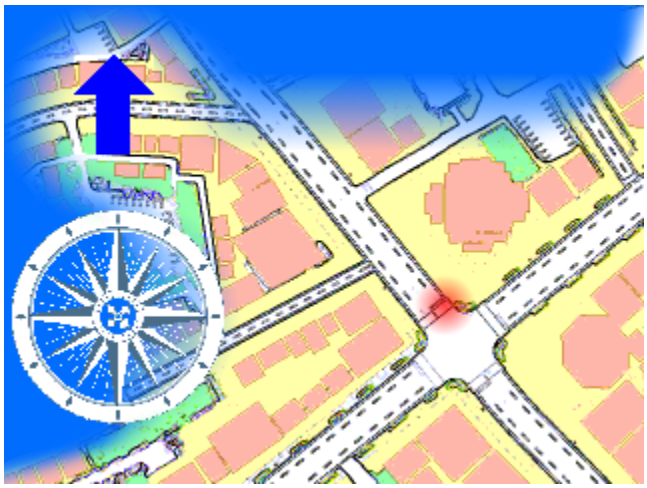
```
// complete the windows params
winprop.width  =  dispParams.xResolution;  // horizontal  display resolution
winprop.height  =  dispParams.yResolution;  // vertical  display  resolution
UTIL_SUCCESS(ret, mmlGdcDispWinCreate(display, &winprop, &win));
UTIL_SUCCESS(ret, mmlGdcDispWinSetSurface(win, MML_GDC_DISP_BUFF_TARGET_COLOR_BUFF,
surfStore));
UTIL_SUCCESS(ret, mmlGdcDispWinCommit(win));
```

6.6 Tutorial: Display Basic

6.6.1 Description

This example realizes an animated but very simple navigation demo done just by the use of layer properties and operations. The focus of this tutorial is to introduce different layer properties and how to set them up.

Figure 6-5 Expected result



6.6.1.1 Learning Goals

The following techniques and features are used:

- Blend 4 layers with pixel based alphas and different color formats.
- Fade a layer.
- Move a layer.
- Switch buffers.
- Use multi layer feature.

6.6.1.2 Layer overview

The example uses the following 4 surfaces:

Layer	Surface name	Preview	Dimension	Color Format	Shown features
LAYER 0	sMap		1024 * 1024	8 bits per pixel RGB (332)	– The surface will be moved in a way that only a part of it is visible. – Sub-pixel precise movements for a smooth animation. – The unusual format (3 bit red, 3 bit green, 2 bit blue) realizes an acceptable memory requirement. ***Note:*** *Compression cannot be used for layers if the whole frame is not inside the display.*
LAYER 1	sFrame		320 * 240	16 bits per pixel RGBA (R3G3B4A6)	– The unusual format (3 bit red, 3 bit green, 4 bit blue, 6 bit alpha) realizes an acceptable memory requirement. ***Note:*** *Compression cannot be used for this window because it is multi layer window.*
LAYER 2	sPosition		32 * 32	32 bit per pixel RGBA	– The layer will be faded in and out.
LAYER 3	sArrow, sArrow_l, sArrow_r		about 40 * 50	1 and 2 bit per pixel Alpha	– The image does not include any color data but only transparency.

The limited number of layer blend units requires using the multi layer feature. It means we use one "normal" window as background window showing the moving map. The frame, the arrow and the position will be realized as multi layer windows, which means that they cannot be blended to each other but they can be blended to the map background window using different properties. The multi layer windows can overlap too like in this example however only the top most window color will be fetched and used for blending.

6.6.2 Chapters

1. Code Description
2. Map Layer
3. Frame Layer
4. Position Layer
5. Arrow Layer

6.6.3 Code Description

We start with driver initialization and setup of the display. The macro UTIL_SUCCESS used in this example is a simple error handling helper.

```
/* Initialize the driver */
UTIL_SUCCESS(ret,     mmlGdcSysInitializeDriver(0));

UTIL_SUCCESS(ret, utMmanReset() );
```

```
/* Allocate some of VRAM for Instruction buffer for the command sequencer. Note, that mmlGdcVideoAlloc is
an application defined routine to manage the VRAM space.
The 2D core driver does not include any management of the VRAM. */
vInstrBuffer = mmlGdcVideoAlloc(fifo_size, 0, NULL);
UTIL_SUCCESS(ret, mmlGdcSysSetInstructionBuffer(vInstrBuffer, fifo_size));

/* Setup and enable the display */
UTIL_SUCCESS(ret, mmlGdcDispOpenDisplay(&dispParams, &display));
```

Reset the surfaces to apply default values.

```
UTIL_SUCCESS(ret, mmlGdcSmResetSurfaceObject(sFrame));
UTIL_SUCCESS(ret, mmlGdcSmResetSurfaceObject(sMap));
UTIL_SUCCESS(ret, mmlGdcSmResetSurfaceObject(sPosition));
UTIL_SUCCESS(ret, mmlGdcSmResetSurfaceObject(sArrow));
```

We load the surfaces for the example using a utility function. The utility function sets all the related properties including compression parameter.

```
/* First we load the the map surface with 1024 * 1024 pixel resolution.
Of course we will read only a part if we use it as layer in this example. */
UTIL_SUCCESS(ret, utSurfLoadBitmap(sMap, map2d, MM_FALSE));

/* Now we load the blue frame surface. Please note that this surface is run length encoded. */
UTIL_SUCCESS(ret, utSurfLoadBitmap(sFrame, frame, MM_FALSE));

/* Next we load a position indicator bitmap */
UTIL_SUCCESS(ret, utSurfLoadBitmap(sPosition, position, MM_FALSE));

/* Finally we load the arrow bitmaps (1 bpp and 2 bpp alpha channel) */
UTIL_SUCCESS(ret, utSurfLoadBitmap(sArrow, arrow, MM_FALSE));
```

Additional we have to create 4 windows:

```
/* create 4 windows for the layer */
//sMap
winprop.topLeftX = 0;
winprop.topLeftY = 0;
winprop.width = dispParams.xResolution; // horizontal display resolution
winprop.height = dispParams.yResolution; // vertical display resolution
winprop.features = MML_GDC_DISP_FEATURE_DECODE; /* We do not need decode.
    However it ensures the driver uses this fetch and not the multilayer fetch.
```

```
        Other way: just open this window as the last one. */
winprop.layerId     = MML_GDC_DISP_LAYER_0; // use layer 0
UTIL_SUCCESS(ret, mmlGdcDispWinCreate(display, &winprop, &wMap));

//sFrame
winprop.topLeftX = 0;
winprop.topLeftY = 0;
winprop.width       = dispParams.xResolution; // horizontal display resolution
winprop.height      = dispParams.yResolution; // vertical display resolution
winprop.features = MML_GDC_DISP_FEATURE_MULTI_LAYER; // use multi layer feature to get more
windows
winprop.layerId     = MML_GDC_DISP_LAYER_1; // use layer 1
winprop.sub_layerId = MML_GDC_DISP_SUB_LAYER_DEFAULT; // sub layer default means the driver will
assign the sub window order. The first opened window is the bottom most.
UTIL_SUCCESS(ret, mmlGdcDispWinCreate(display, &winprop, &wFrame));

//sPosition
winprop.topLeftX = CENTER_X - 16;
winprop.topLeftY = CENTER_Y - 16;
winprop.width       = 32;
winprop.height      = 32;
winprop.features = MML_GDC_DISP_FEATURE_MULTI_LAYER;
winprop.layerId    =    MML_GDC_DISP_LAYER_1;
UTIL_SUCCESS(ret, mmlGdcDispWinCreate(display, &winprop, &wPosition));

//sArrow
winprop.topLeftX    = 30;
winprop.topLeftY    = 25;
winprop.width        = 50;
winprop.height       = 50;
winprop.features    = MML_GDC_DISP_FEATURE_MULTI_LAYER;
winprop.layerId     = MML_GDC_DISP_LAYER_1;
UTIL_SUCCESS(ret,    mmlGdcDispWinCreate(display,    &winprop,    &wArrow));
```

The following sections describe how these surfaces are used in this example.

6.6.4 Map Layer

The map will be assigned to the wMap window:

```
UTIL_SUCCESS(ret, mmlGdcDispWinSetSurface(wMap,
MML_GDC_DISP_BUFF_TARGET_COLOR_BUFF, sMap));
```

Besides this we want to see a moving map. That means we have to move the layer in our animation loop. The driver ensures that only pixels inside the screen of this surface are read from memory.

```
GetPosition(frameCount, &x, &y, &winker);

/* To get a moving map we have to recalculate a new matrix for the map layer. */
utMat3x2LoadIdentity(mat_geo);

/* Move it in a way that the requested center point fits the current position */
utMat3x2Translate(mat_geo, (MM_FLOAT)(CENTER_X - x), (MM_FLOAT)(CENTER_Y - y));

/* Assign the matrix to the window */
UTIL_SUCCESS(ret,    mmlGdcDispWinSetMatrix(wMap,
MML_GDC_DISP_BUFF_TARGET_COLOR_BUFF, mat_geo));
UTIL_SUCCESS(ret,       mmlGdcDispWinCommit(wMap));
```

6.6.5 Frame Layer

The frame layer is the simplest layer for this example because it is not included in an animation:

```
UTIL_SUCCESS(ret, mmlGdcDispWinSetSurface(wFrame,
MML_GDC_DISP_BUFF_TARGET_COLOR_BUFF, sFrame));
UTIL_SUCCESS(ret, mmlGdcDispWinSetBlendMode(wFrame,
MML_GDC_DISP_BLEND_SOURCE_ALPHA |
MML_GDC_DISP_BLEND_SOURCE_MULTIPLY_ALPHA));
UTIL_SUCCESS(ret, mmlGdcDispWinSetAttribute(wFrame, MML_GDC_DISP_WIN_ATTR_COLOR,
0x80FFFFFF));
UTIL_SUCCESS(ret, mmlGdcDispWinCommit(wFrame));
```

The related properties required for the compression were already assigned to the surface in the utSurfLoadBitmap function. In this case the RLA compression is used because it shrinks the size for this bitmap to 12.0% of the original size.

6.6.6 Position Layer

The position layer demonstrates the fading capabilities of the hardware. To fade a layer with pixel based alpha information, the following calculation inside the hardware is required:

Alpha = Alpha pix * Alpha fade

For the calculation of the Alpha value we have to assign the related properties to the wPosition window:

```
UTIL_SUCCESS(ret, mmlGdcDispWinSetSurface(wPosition,
MML_GDC_DISP_BUFF_TARGET_COLOR_BUFF,     sPosition));
/* We want to fade this layer: multiply pixel-alpha * const alpha. */
UTIL_SUCCESS(ret, mmlGdcDispWinSetBlendMode(wPosition,
MML_GDC_DISP_BLEND_GLOBAL_ALPHA | MML_GDC_DISP_BLEND_SOURCE_ALPHA |
MML_GDC_DISP_BLEND_SOURCE_MULTIPLY_ALPHA));
UTIL_SUCCESS(ret, mmlGdcDispWinSetAttribute(wPosition, MML_GDC_DISP_WIN_ATTR_COLOR,
0xFF0000FF));
```

To realize the blink effect we have to modify the color:

```
UTIL_SUCCESS(ret, mmlGdcDispWinSetAttribute(wPosition, MML_GDC_DISP_WIN_ATTR_COLOR,
blink));
UTIL_SUCCESS(ret, mmlGdcDispWinCommit(wPosition));
```

6.6.7 Arrow Layer

The arrow layer is a one bit alpha mask only. So we have to define a constant color for the missing color data. In addition we enable the pre-multiplication of color and alpha because the default layer blend mode expects a pre-multiplied image and we have a constant color only.

```
UTIL_SUCCESS(ret, mmlGdcDispWinSetBlendMode(wArrow,
MML_GDC_DISP_BLEND_SOURCE_ALPHA |
MML_GDC_DISP_BLEND_SOURCE_MULTIPLY_ALPHA));
UTIL_SUCCESS(ret, mmlGdcDispWinSetAttribute(wArrow, MML_GDC_DISP_WIN_ATTR_COLOR,
0x0000FFFF));
```

In the animation loop we simple change the arrow:

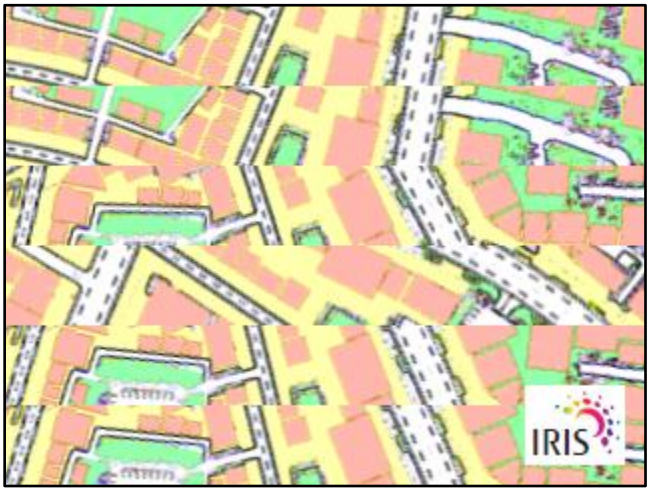
```
switch(winker)
{
    case -1: surfArrow = sArrow_l; break;
    case  0: surfArrow = sArrow; break;
    case  1: surfArrow = sArrow_r; break;
}

/* Some 2D core drivers use layer rotation at this position to animate the arrow.
```

```
We cannot use simple rotation while using sub-windows but we can change the image.
So the following matrix calculation just moves the surface to the window center.
*/
utMat3x2LoadIdentity(mat_geo);
utMat3x2Translate(mat_geo, 25.0f, 25.0f);
utMat3x2Translate(mat_geo, (- (MM_FLOAT)utSurfWidth(sArrow) * 0.5f), (-
(MM_FLOAT)utSurfHeight(sArrow) * 0.5f));
/* Set matrix */
UTIL_SUCCESS(ret,     mmlGdcDispWinSetMatrix(wArrow,
MML_GDC_DISP_BUFF_TARGET_COLOR_BUFF, mat_geo));
/* Set new surface */
UTIL_SUCCESS(ret,    mmlGdcDispWinSetSurface(wArrow,
MML_GDC_DISP_BUFF_TARGET_COLOR_BUFF, surfArrow));
UTIL_SUCCESS(ret,    mmlGdcDispWinCommit(wArrow));
```

6.7 Tutorial: Display_Extended

6.7.1 Description

Figure 6-6 Display Extended



The focus of Display_Extended is the synchronization of blit and buffer swap operations.

- Open multiple windows on the display and prepare double buffering.
- Trigger simple render operations and buffer swaps for each window.
- Use different swap intervals for windows.

6.7.2 Setup

The initialization routine opens the display controller. Then it opens any vertical arranged windows. Each window prepares

- 2 frame buffers that will be used as foreground and background buffer.
- A blit context.
- And a sync object.

A structure for each window keeps all important variables to control the window:

```
struct    DOUBLE_BUFFERED_WINDOW{
    MML_GDC_DISP_WINDOW win; // the window handle
    MML_GDC_SURFACE_CONTAINER sFramebuffer[2]; // two buffers described in surface objects.
    MML_GDC_SYNC_CONTAINER sync; // a sync object used
    MML_GDC_PE_CONTEXT_CONTAINER ctx; // context for drawing
    MM_U08 id; // an id storing which buffer is currently the foreground buffer
    MM_FLOAT fRot; // a draw related parameter
};
```

The final step for each window is getting a sync object of the window pipe. This sync object can be used to detect if the OpenWindow call is finished in the HW.

6.7.3 Draw function

The main draw function calls a draw for each window. Each window draw function checks first the window sync object. If the sync object signals a timeout the function returns.

```
ret = mmlGdcSyncWait(&pdbWin->sync, 0);
if (ret == MML_ERR_GDC_SYNC_TIMEOUT)
    return MML_OK;
```

Using this mechanism the drawing loop will not consume CPU time if the previous buffer swap is not yet finished. The next step is rendering the new frame in the back buffer. All these render operations will be pushed in the

command sequencer queue and executed sequential by the hardware. So if we now assign the new buffer to the window it is possible that the new buffer becomes visible before rendering is finished.

To avoid this it is possible to poll the end of the blit operation using mmlGdcPeFinish(). A better way is to use a sync object:

```
UTIL_SUCCESS(ret, mmlGdcPeSync(&pdbWin->sync));
UTIL_SUCCESS(ret, mmlGdcDispWinWaitSync(pdbWin->win, &pdbWin->sync));
UTIL_SUCCESS(ret, mmlGdcDispWinSetSurface(pdbWin->win,
MML_GDC_DISP_BUFF_TARGET_COLOR_BUFF, &pdbWin->sFramebuffer[pdbWin->id] ));
UTIL_SUCCESS(ret, mmlGdcDispWinCommit(pdbWin->win));
```

It requests a sync object from the pixel engine and pushes it to the window pipe before the new buffer is assigned to the window. All these functions are non blocking for the CPU and the driver will ensure that the hardware will be triggered in the correct order.

6.7.4 Swap interval

The windows are set to different swap intervals:

```
UTIL_SUCCESS(ret, mmlGdcDispWinSetAttribute(s_dbw[i].win,
MML_GDC_DISP_WIN_ATTR_SWAP_INTERVAL, window_assignment[i].swap_interval));
```

This feature can be used to control the window refresh interval. Very important windows may keep the default swap interval 1 but low priority windows with may be GPU consuming draw operations can be set to a swap interval 2 or 3. In this case the window will be updated with 30 Hz or 20 Hz for a display with 60 Hz refresh rate.

6.8 Tutorial: Speedometer

6.8.1 Summary

This example realizes a simple speedometer. The aim is to use 2 layers:

- One as a static background for the scale.
- And one dynamic layer with a rotating needle and a fixed hub around the rotation center. The hub image has a light shadow and must not be rotated with the needle.

The user can switch between 4 possible drawing versions which are commented on later, by pressing the "right" button. By pressing the "left" button, the bShowDrawRects property can be toggled, which draws different rectangles to visualize the drawing areas.

The sample uses a "single buffer render mode". However different to the Chart sample this demo uses only the blanking period of the panel timing. That’s why it is important to use very fast render operations.

Figure 6-7 Expected result

6.8.2 Learning Goals

The following techniques and features are used:

- Show different ways to restore and render the needle layer.
- Usage of mmlGdcPeSelectArea, mmlGdcPeActiveArea and mmlGdcPeGetDrawBox.
- Show the coordinate system transformation.
- Use a colored 4 bit per pixel layer.

6.8.3 Chapters

1. Preparation
2. Matrix operations to scale, rotate and translate images
3. Show different versions to restore and draw the needle layer

6.8.4 Preparation

First step is to initialize the driver and setup the display:

```
/* Initialization of driver and display */
/* Initialize the driver */
UTIL_SUCCESS(ret, mmlGdcSysInitializeDriver(0));
UTIL_SUCCESS(ret, utMmanReset() );

/* Allocate some of VRAM for Instruction buffer for the command sequencer. Note, that mmlGdcVideoAlloc is an application defined routine to manage the VRAM space. The 2D core driver does not include any management of the VRAM. */
vInstrBuffer  =  mmlGdcVideoAlloc(fifo_size, 0, NULL);
UTIL_SUCCESS(ret, mmlGdcSysSetInstructionBuffer(vInstrBuffer, fifo_size));

/* Setup and enable the display */
UTIL_SUCCESS(ret, mmlGdcDispOpenDisplay(&dispParams, &s_display));
```

If bShowDrawRects is set, we prepare an additional layer sComment. This layer represents a minimal colored layer: only one bit is reserved for each color channel and for alpha. As the utility function utSurfCreateBuffer only supports common color formats, we create our own function CreateCommentSurface.

```
static  MM_ERROR  CreateCommentSurface(MML_GDC_SURFACE  sComment)
{
    MM_ERROR ret = MML_OK;
    void *vp;

    vp = mmlGdcVideoAlloc( (BGR_WIDTH * BGR_HEIGHT * 4 / 8), 0, NULL);
    if (vp == NULL)
    {
        return MML_ERR;
    }
    UTIL_SUCCESS(ret, mmlGdcSmResetSurfaceObject(sComment));
    mmlGdcSmSetAttribute(sComment, MML_GDC_SURF_ATTR_WIDTH, BGR_WIDTH);
    mmlGdcSmSetAttribute(sComment, MML_GDC_SURF_ATTR_HEIGHT, BGR_HEIGHT);
    mmlGdcSmSetAttribute(sComment, MML_GDC_SURF_ATTR_BITPERPIXEL, 4);
    mmlGdcSmSetAttribute(sComment, MML_GDC_SURF_ATTR_COLORBITS, 0x01010101);
    mmlGdcSmSetAttribute(sComment, MML_GDC_SURF_ATTR_COLORSHIFT, 0x03020100);
```

```
    mmlGdcSmSetAttribute(sComment, MML_GDC_SURF_ATTR_BASE_ADDRESS, (MM_U32)vp);

    return ret;
}
```

The background layer (the scale) will only be visible if bShowDrawRects is not set, to keep the example simple. As this layer is not of interest for the tutorial, we just use a helper function DrawBgr() to draw several image sources in our sBgr buffer.

```
/* Create a surface for background */
UTIL_SUCCESS(ret, mmlGdcSmResetSurfaceObject(&s_sBgr));
UTIL_SUCCESS(ret, utSurfCreateBuffer(&s_sBgr, BGR_WIDTH, BGR_HEIGHT,
MML_GDC_SURF_FORMAT_R5G6B5));
/* draw the scale on background surface. You may use a fixed bitmap too. */
UTIL_SUCCESS(ret, DrawBgr(&s_sBgr, s_sSrc, s_mat));
/* display the background surface on background layer */
UTIL_SUCCESS(ret, mmlGdcDispWinSetSurface(s_winBgr,
MML_GDC_DISP_BUFF_TARGET_COLOR_BUFF, &s_sBgr));
UTIL_SUCCESS(ret, mmlGdcDispWinCommit(s_winBgr));
```

Now we create a layer for the hub and needle. We need a buffer with an alpha channel because the layer blending should only pass the needle and hub. All other parts must have an alpha = 0 value so that they are not visible.

```
/* Create a window for needle layer */
windowProp.layerId   =   MML_GDC_DISP_LAYER_1;
UTIL_SUCCESS(ret, mmlGdcDispWinCreate(s_display, &windowProp, &s_winNeedle));
UTIL_SUCCESS(ret, mmlGdcDispWinSetBlendMode(s_winNeedle,
MML_GDC_DISP_BLEND_SOURCE_ALPHA));

UTIL_SUCCESS(ret, mmlGdcPeResetContext(&s_ctx));

/* Create a target surface for the needle. This is the focus layer for this demonstration. */
UTIL_SUCCESS(ret, mmlGdcSmResetSurfaceObject(&s_sNeedle));
UTIL_SUCCESS(ret, utSurfCreateBuffer(&s_sNeedle, BGR_WIDTH, BGR_HEIGHT,
MML_GDC_SURF_FORMAT_R6G6B6));
```

6.8.5 Matrix operations to scale, rotate and translate images

All geometry changes such as translation, rotation, sharing, scaling and mirroring in the blit path are based on matrix settings. The application can calculate such matrices on its own or by using the utility functions from the driver. The x, y offset in the mmlGdcPeBlt function can be used for simple translations.

The default behavior is that all matrices are reset to identity matrices. That means a

```
mmlGdcPeBlt(&ctx, 10, 20)
```

would copy the source buffer to the target buffer with an offset x = 10 and y = 20. Depending on the MML_GDC_PE_ATTR_ZERO_POINT settings the y offset is counted from the upper or lower left store surface coordinate.

An equivalent operation with a matrix would be the following if sSrc is the source surface.

```
Mat3x2LoadIdentity(mat);
Mat3x2Translate(mat, 10, 20);
mmlGdcPeSetMatrix(ctx, MML_GDC_PE_SRC, mat);
mmlGdcPeBlt(0, 0);
```

However there are differences if several source buffers are involved. If the offset x, y is represented by a matrix.

$$\mathrm{Moffs} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The following relationship to the store surface is be valid for the SRC and MASK surface (represented by ’Ms’):

$$\begin{pmatrix} \mathrm{Xstore} \\ \mathrm{Ystore} \end{pmatrix} = \mathrm{Moffs} \times \mathrm{Ms} \times \begin{pmatrix} \mathrm{Xs} \\ \mathrm{Ys} \end{pmatrix}$$

The path for the DST calculation is a little bit different (Mdst is the DST matrix):

$$\begin{pmatrix} \mathrm{Xstore} \\ \mathrm{Ystore} \end{pmatrix} = \mathrm{Mdst} \times \begin{pmatrix} \mathrm{Xdst} \\ \mathrm{Ydst} \end{pmatrix}$$

This means the offsets are valid for all blit paths except the DST and the individual matrix for each source buffer is used for this path only.

This behavior can be used to simplify any operations. For instance, you can set a mirror matrix to the store and without any other changes you can mirror all blit operations for this target.

In the speedometer example, we calculate matrices for the rotation center of the images and use the blit offset to move it to the correct position.

```
mmlGdcPeBlt(&ctx, BGR_WIDTH * 0.5f, ROT_CENTER_Y);
```

All source surfaces including hub get a similar matrix (except background) in PrepareSurfaces.

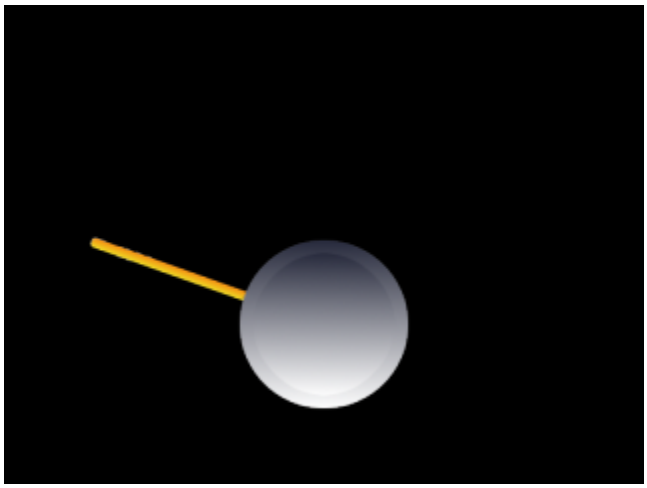
```
utMat3x2LoadIdentity(mat[0]);

/* we have 7 sources so we can simply handle it in an array. */
for (i = 1; i < 7; i++)
{
    UTIL_SUCCESS(ret, mmlGdcSmResetSurfaceObject(&sSrc[i]));
    UTIL_SUCCESS(ret, utSurfLoadBitmap(&sSrc[i], mysrc[i].name, MM_FALSE));

    // prepare matrix array for surfaces
    utMat3x2LoadIdentity(mat[i]);
    // align the rotation centers of surfaces
    utMat3x2Translate(mat[i], -(MM_FLOAT)
    utSurfWidth(&sSrc[i]) * 0.5f, -mysrc[i].fCenterY);
}
```

The rotation angle is changed frame by frame, so we have to calculate a new matrix each time for this surface. We encapsulated it in the function GetRotMatrix:

```
static MM_ERROR GetRotMatrix(MM_U32 SurfID, MML_GDC_SURFACE_CONTAINER *sSrc,
MM_FLOAT fAngle, Mat3x2 *mat)
{
    MM_ERROR ret = MML_OK;

    // move the surface to the rotation center
    utMat3x2LoadIdentity(mat[SurfID]);
    utMat3x2Translate(mat[SurfID], (MM_FLOAT)BGR_WIDTH * 0.5f, (MM_FLOAT)ROT_CENTER_Y);
    utMat3x2Rot(mat[SurfID], fAngle);
    utMat3x2Translate(mat[SurfID], -(MM_FLOAT)
    utSurfWidth(&sSrc[SurfID]) * 0.5f, -mysrc[SurfID].fCenterY);
    return ret;
}
```

6.8.6 Show different versions to restore and draw the needle layer

Figure 6-8 Previous layer frame



As mentioned, we want to discuss different possibilities. The scenario should be always the same: a previous frame of the sNeedle surface was drawn and the new needle position must be drawn instead.

- Version 1
- Version 2
- Version 3
- Version 4

6.8.6.1 Version 1

A typical draw loop clears the buffer and draws the new objects on it. We perform 3 rendering steps:

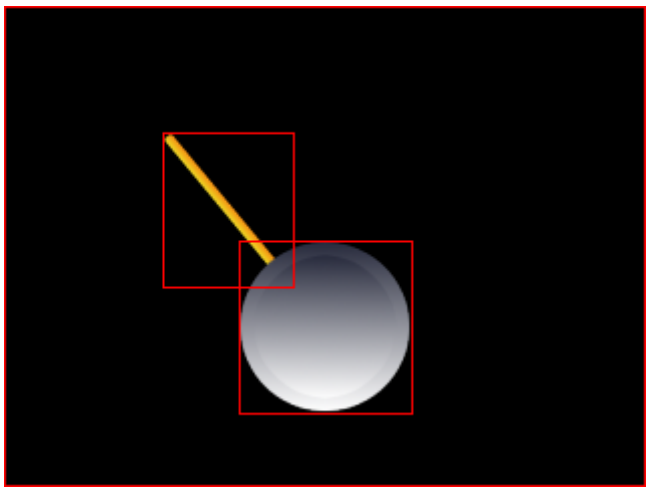
- The fill instruction clears the whole buffer.
- Next the rotated needle is drawn.
- Finally the hub is drawn.

```
// Set target to sNeedle surface and enable blending with MML_GDC_PE_DST
UTIL_SUCCESS(ret, mmlGdcPeBindSurface(&ctx, MML_GDC_PE_STORE | MML_GDC_PE_DST, &sNeedle));

while(TRUE)
{
    // Clear the last frame
    UTIL_SUCCESS(ret, mmlGdcPeColor(&ctx, 0, 0, 0, 0));
    UTIL_SUCCESS(ret, mmlGdcPeFill(&ctx, 0, 0, BGR_WIDTH, BGR_HEIGHT));

    // Draw the rotated needle
    UTIL_SUCCESS(ret, mmlGdcPeBindSurface(&ctx, MML_GDC_PE_SRC, &sSrc[BMP_NEEDLE]));
    UTIL_SUCCESS(ret, mmlGdcPeSetMatrix(&ctx, MML_GDC_PE_SRC, MML_GDC_PE_GEO_MATRIX_FORMAT_3X2, mat[BMP_NEEDLE]));
    UTIL_SUCCESS(ret, mmlGdcPeBlt(&ctx, 0, 0));
```

```
    // Draw the hub
    UTIL_SUCCESS(ret, mmlGdcPeBindSurface(&ctx, MML_GDC_PE_SRC, &sSrc[BMP_HUB]));
    UTIL_SUCCESS(ret, mmlGdcPeSetMatrix(&ctx, MML_GDC_PE_SRC,
MML_GDC_PE_GEO_MATRIX_FORMAT_3X2, mat[BMP_HUB]));
    UTIL_SUCCESS(ret, mmlGdcPeBlt(&ctx, BGR_WIDTH * 0.5f, ROT_CENTER_Y));
}
```

The next image shows the draw boxes for these 3 rendering steps:

Figure 6-9 Version 1



6.8.6.2 Version 2

In the previous implementation the store buffer is read twice, first when blending the needle, then again when blending the hub on top of it. To avoid this additional memory access, we can blend both sources in one step onto the store buffer. The problem: by default the driver only processes the bounding box of the source buffer. In our example, the hub must be blended over the needle so just a part of the needle would be visible. To avoid this issue we can force the driver to process both the SRC and the DST frame buffer by using the mmlGdcPeSelectArea function.

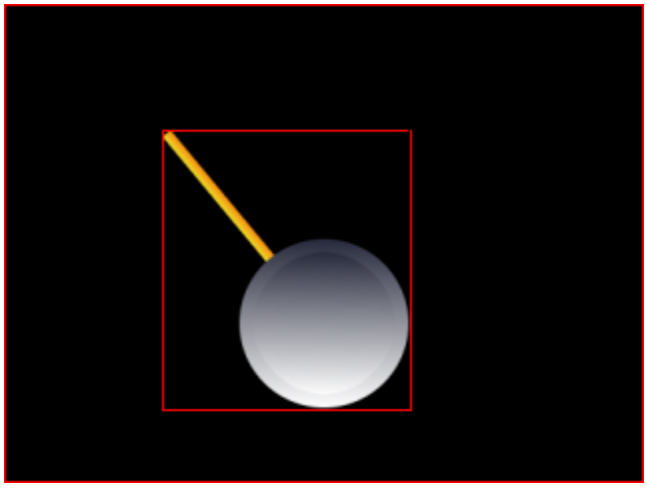
```
/* Here we blend hub over rotated needle to store */
UTIL_SUCCESS(ret, mmlGdcPeBindSurface(&ctx, MML_GDC_PE_STORE, &sNeedle));
UTIL_SUCCESS(ret, mmlGdcPeBindSurface(&ctx, MML_GDC_PE_DST, &sSrc[BMP_NEEDLE]));
UTIL_SUCCESS(ret, mmlGdcPeBindSurface(&ctx, MML_GDC_PE_SRC, &sSrc[BMP_HUB]));
UTIL_SUCCESS(ret, mmlGdcPeSetMatrix(&ctx, MML_GDC_PE_SRC,
MML_GDC_PE_GEO_MATRIX_FORMAT_3X2, mat[BMP_HUB]));

/* We have to render the combined bounding box of needle and hub in this case */
UTIL_SUCCESS(ret, mmlGdcPeSelectArea(&ctx, MML_GDC_PE_SRC | MML_GDC_PE_DST));

while(TRUE)
{
```

```
    /* Clear the last frame */
    UTIL_SUCCESS(ret, mmlGdcPeColor(&ctx, 0, 0, 0, 0));
    UTIL_SUCCESS(ret, mmlGdcPeFill(&ctx, 0, 0, BGR_WIDTH, BGR_HEIGHT));
    /* Blend the hub over rotated needle */
    UTIL_SUCCESS(ret, mmlGdcPeSetMatrix(&ctx, MML_GDC_PE_DST,
MML_GDC_PE_GEO_MATRIX_FORMAT_3X2, mat[BMP_NEEDLE]));
    UTIL_SUCCESS(ret, mmlGdcPeBlt(&ctx, BGR_WIDTH * 0.5f, ROT_CENTER_Y));
}
```

This time only 2 rendering steps are required:

- The fill instruction clears the whole buffer.
- Blend the hub over the rotated needle. The driver will calculate and render the bounding box of the rotated needle and the hub.

Figure 6-10 Version 2



6.8.6.3 Version 3

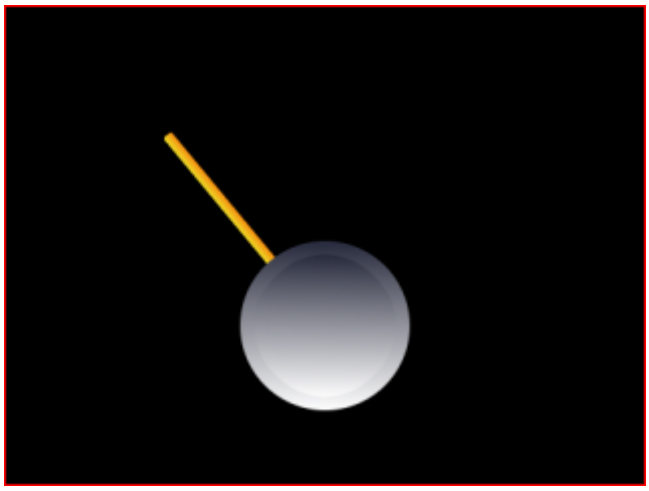
Can we use only one rendering pass by rendering images that are larger than the source? We can! We just define that the rendering rectangle is defined by the target buffer.

```
/* Again blend hub over rotated needle to store ... */
UTIL_SUCCESS(ret, mmlGdcPeBindSurface(&ctx, MML_GDC_PE_STORE, &sNeedle));
UTIL_SUCCESS(ret, mmlGdcPeBindSurface(&ctx, MML_GDC_PE_DST, &sSrc[BMP_NEEDLE]));
UTIL_SUCCESS(ret, mmlGdcPeBindSurface(&ctx, MML_GDC_PE_SRC, &sSrc[BMP_HUB]));
UTIL_SUCCESS(ret, mmlGdcPeSetMatrix(&ctx, MML_GDC_PE_SRC,
MML_GDC_PE_GEO_MATRIX_FORMAT_3X2, mat[BMP_HUB]));

/* ... but we render the whole target buffer in one step. */
UTIL_SUCCESS(ret,     mmlGdcPeSelectArea(&ctx, MML_GDC_PE_STORE));

while(TRUE)
{
```

```
    /* Blend the hub over rotated needle but we draw the whole target frame so we don't need to clear the buffer */
    UTIL_SUCCESS(ret, mmlGdcPeSetMatrix(&ctx, MML_GDC_PE_DST, MML_GDC_PE_GEO_MATRIX_FORMAT_3X2, mat[BMP_NEEDLE]));
    UTIL_SUCCESS(ret, mmlGdcPeBlt(&ctx, BGR_WIDTH * 0.5f, ROT_CENTER_Y));
}
```

Now we have only one rendering step. The hardware fills black pixels outside the hub and needle buffer, and this is exactly what we need to clear the previous frame.

Figure 6-11 Version 3



6.8.6.4 Version 4

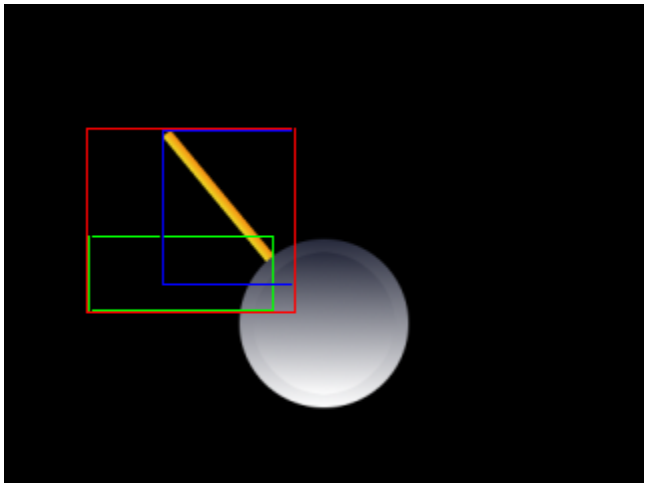
The previous version must always update the whole layer frame although only a very small part (the old needle) must be redrawn. The most efficient way would be to re-render only the new and the old needle parts.

```
UTIL_SUCCESS(ret, mmlGdcPeBindSurface(&ctx, MML_GDC_PE_STORE, &sNeedle));
/* Blend hub over rotated needle to store. */
UTIL_SUCCESS(ret, mmlGdcPeBindSurface(&ctx, MML_GDC_PE_DST, &sSrc[BMP_NEEDLE]));
UTIL_SUCCESS(ret, mmlGdcPeBindSurface(&ctx, MML_GDC_PE_SRC, &sSrc[BMP_HUB]));
UTIL_SUCCESS(ret, mmlGdcPeSetMatrix(&ctx, MML_GDC_PE_SRC, MML_GDC_PE_GEO_MATRIX_FORMAT_3X2, mat[BMP_HUB]));

/* Needle and store define the bounding box. */
UTIL_SUCCESS(ret, mmlGdcPeSelectArea(&ctx, MML_GDC_PE_DST | MML_GDC_PE_STORE));

while(TRUE)
{
    UTIL_SUCCESS(ret, mmlGdcPeSetMatrix(&ctx, MML_GDC_PE_DST, MML_GDC_PE_GEO_MATRIX_FORMAT_3X2, mat[BMP_NEEDLE]));
```

```
    /* Blend the hub over rotated needle */
    UTIL_SUCCESS(ret, mmlGdcPeBlt(&ctx, (MM_FLOAT)BGR_WIDTH * 0.5f,
(MM_FLOAT)ROT_CENTER_Y));

    /* Determine draw box for the current frame ... */
    UTIL_SUCCESS(ret, mmlGdcPeGetDrawBox(&ctx, &x, &y, &w, &h, MM_TRUE));

    /* ... and assign it as active area to the target for the next frame. This box includes the current needle
and must be repainted in the next frame */
    UTIL_SUCCESS(ret,     mmlGdcPeActiveArea(&ctx, MML_GDC_PE_STORE, x, y, w, h));
}
```

Again we have only one rendering step, but this time the rendering box (red) is much smaller. It is the bounding box from the previously rendered needle (green) and the new needle box (blue). The mmlGdcPeGetDrawBox returns the drawing box of the last rendering step and this box is set as the ActiveArea for the store surface. Please note that mmlGdcPeGetDrawBox does not include the drawing box of the store surface, otherwise the box would be increased with each new frame.

Figure 6-12 Version 4

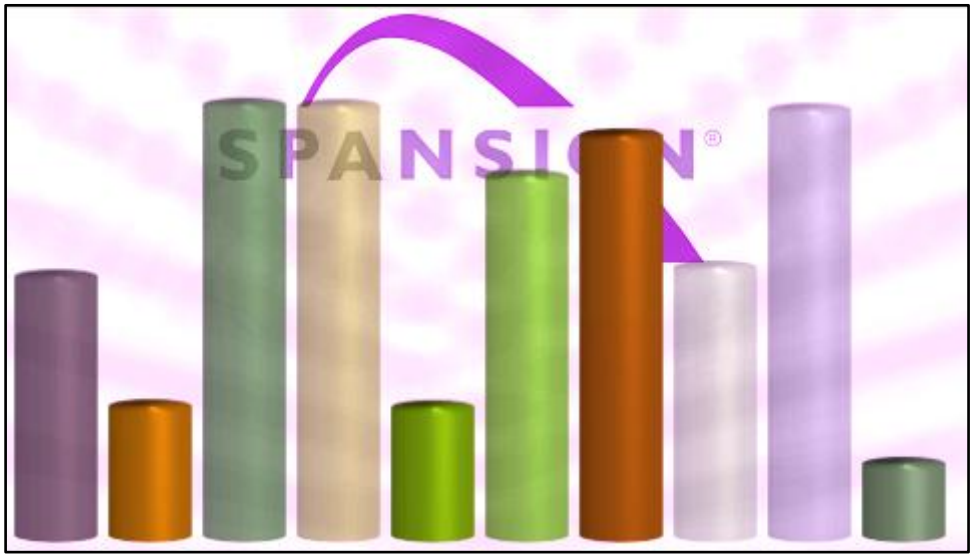


6.9 Tutorial: Chart (Single render buffer sample)

6.9.1 Summary

This example shows an animated chart using a single buffer render mode. Source code: 04_sample/basic_graphics/chart/.

Figure 6-13 chart



6.9.1.1 Learning Goals

The following techniques and features are used:

- Work with clip rectangle for the STORE surface.
- Synchronize display read and blit operations.
- Analyze render time.
- Analyze command sequencer buffer size.
- Color matrix operations for blit operations.
- Use alpha multiply with MASK surface.

6.9.1.2 Memory Calculation for VRAM

The target device has a VRAM size of 512 kByte. The panel used for our samples has a size of 480 ∗ 272 pixels. The sample should use a high quality render buffer requiring an alpha channel. If we want to use at least 6 bit for all color and the alpha channel we need 480 ∗ 272 ∗ 24 ∗ 2 / 8 = 765 kByte.

That means double buffering is not possible for such a resolution and color format. To render such targets anyway it is possible to use a single buffer render mode. In single buffer mode we need only 383 kByte to store the frame buffer.

The command sequencer size assigned by mmlGdcSysSetInstructionBuffer() must be big enough to store all operations for one render loop. This is important because the blit operations must be queued until the display controller passes a defined line in a single buffer environment. The example starts with a Command Sequencer buffer size measurement for the first frame if MEASURE_CMD_SEQ is defined with the following result:

CmdSeq buffer size: 13.46 of 16.00 kB used

The sample draws 10 bars, 2 background images and a debug bar in one half of the frame. It means the driver needs approximately 500 Byte (or 125 registers writes) for one blit in this sample. Please note, the required bytes for a blit depends on the operation and properties. We need 3 sources to render the bars. In many cases only 2 sources are used. However, a blit with a surface with an index table of 256 colors needs much more instruction buffer because the palette with 256 ∗ 4 register values already requires 1 kB command sequencer space.

To allow a fancy background animation the init function also allocates a 128∗128∗4 (alpha) bpp surface and renders the following pattern in this surface:

Figure 6-14 Background pattern

All these memory blocks together need about 411 kB VRAM.

6.9.1.3 Render time analysis

To generate a render job that generates a real GPU load for the 2D render hardware this sample uses a background animation using 2 bilinear full screen rotations. For real applications this is not a typical use case and the GPU load will be smaller. The animated bars of the chart are the foreground for this sample. Approximately up to 75% of the screen size are filled by the bars. It means that the buffer will be filled about 2.75 times by bilinear blit operations. Bilinear blits (here rotation and sub-pixel movement) need 2 clock cycles for one pixel so we expect 480∗272∗2∗2.75 clock cycles to render one frame. For a 160 MHz clocked blit engine it requires about 5 ms or 1/3 frame if we use a 60 Hz panel. In practice the render time will be much longer because the cylinderical bitmap used to blit the bars will be read from external flash and access to external resources are not as fast as VRAM access. To see the render time of about 5 ms you can simple change the line

```
UTIL_SUCCESS(ret,     s_sCylinder.SurfLoadBitmap(cylinder));
```

to

```
UTIL_SUCCESS(ret,    s_sCylinder.SurfLoadBitmap(cylinder,    MM_TRUE));
```

This change copies the bitmap to VRAM with fast access time. The example does not make a VRAM copy to generate the high GPU load.

6.9.1.4 Single buffer render mode

One possible implementation for single buffer render mode is to use the blanking period of the timing only. This procedure is used in the Speedometer sample.

In many cases the banking period is too short to redraw all animated buffers. In this case the application can force the HW only to update a part of the frame buffer, if this part of the frame buffer is currently not read by the display controller.

This example uses a single full screen buffer, splitted into an upper and a lower part. It is also possible to divide the screen resolution by using smaller windows. Using different windows is the preferred version because it allows to render a window one time per frame. However, the size and position of the windows must fit the memory scan order for the panel. Typical panels are landscape panels that means the windows must be arranged vertical.

For a single buffer window solution the render function must ensure that only parts of the buffer a updated. This can be realized by using the STORE clipping function of the driver:

```
/* To use partial rendering we switch on clipping and set the rectangle */
mmlGdcPeSurfAttribute(s_ctx, MML_GDC_PE_STORE, MML_GDC_PE_SURF_ATTR_USE_CLIPPING, MM_TRUE);
mmlGdcPeActiveArea(s_ctx, MML_GDC_PE_STORE, 0, y_start, s_win.GetWidth(), lines);
```

This code ensures that all mmlGdcPeBlt() calls never write pixels to the target buffer below line y_start or above y_start + lines. In some cases the driver will detect that a blit operation for the upper part does not affect any pixels in the clipped target buffer. The driver will generate a warning in this case and does not trigger any operation in the HW.

Note:

- *Please remember the default zero point for blit operations is the lower left corner of the buffer.*

To render the whole frame we need to call our render function 2 times and add the required instructions for synchronization:

```
UTIL_SUCCESS(ret, mmlGdcPeWaitForDispFrameEnd(s_display, s_nSyncPoint));
UTIL_SUCCESS(ret, Render(s_display.GetHeight() - s_nSyncPoint, s_nSyncPoint));
UTIL_SUCCESS(ret, mmlGdcPeWaitForDispFrameEnd(s_display, s_display. GetHeight()));
UTIL_SUCCESS(ret, Render(0, s_display.GetHeight() - s_nSyncPoint));
```

The first instruction is a wait instruction for the line position s_nSyncPoint. Then we render the buffer part above the sync point. Now we wait for the end of the screen and start to render the lower part of the screen.

s_nSyncPoint is set to a line below the middle of the screen because the bars are more present in the lower part so this part will take more time.

This example splits the target buffer into 2 parts. It is possible to use 3 or more parts however it is not recommended because each sync point will generate a render gap because the command sequencer must wait for a display controller line.

Note:

- *It is possible to get a frame drop if the render time is too long for one part and the display line sync point is already passed when the command sequencer reaches this instruction. It means the command sequencer will wait one frame until the expected line is passed next time.*

6.9.1.5 Render time visualization

If the render time is critical for a single buffer solution it might be helpful to visualize the render time. The sample application blits for both render parts a red line at different positions on the left side of the screen. Different to all other blits this line will be drawn over the whole surface including the part that is currently read by the display controller. The display will not show the line while the render task is still ongoing. But if the blit queue executes this line drawing operation the display controller will read this new rendered object.

Figure 6-15 Render time analysis

The application supports the keys on the developer board. For this example the up and down keys can be used to modify the line split of the upper and lower part. It can be used to force a wrong splitting and see the render issues. The left key can be used to switch off the background animation. You can see a relaxed render time in that case.

6.9.1.6 Render tricks

The following render tricks are used for this sample:

- Background animation: Two rotated and up scaled buffers with weak alpha value generate the background animation.
- Mask buffer multiplication: To draw the diagram it is necessary to modify the cylinderical bar height. Scaling is not possible because it deforms the 3D-optic. Therefore the sample reads for each cylinderical bar and blit the bitmap twice. The MASK surface only needs the alpha channel of the bitmap. The SRC surface reads the alpha and the color channels of the bitmap but with a vertical offset realized by a geometry matrix operation. The default operation for MASK and SRC surface is alpha multiplication. An additional constant alpha multiplication realizes the semi-transparency of some bars. The result of this product is finally used for the blend operation against the animated background.
- Color modification: The sample uses the color matrix to colorize the gray image source.

6.10 Tutorial: Cover Flow

6.10.1 Summary

This example demonstrates several Pixel Engine features in form of a cover flow. Source code: 04_sample/basic_graphics/coverflow/.

Figure 6-16 coverflow

6.10.2 Usage

Use the "right" button to switch between circle and perspective mode.

6.10.2.1 Learning Goals

The following techniques and features are used:

- Work with matrices
- z-order sorting

The focus in this tutorial is not the initialization nor the double buffer technique.

6.10.2.2 Matrix Calculation

The driver supports an 3∗2 matrix. It allows to translate, rotate, scale and share an bitmap.

However to simplify the development task we decide to make the matrix calculation with a 4∗4 matrix first. This is a matrix format that is well documented because it is used for many OpenGL applications.

A matrix calculation like this for the circle mode can be the following. Please note:

- It is easier to read the matrix operation from bottom to top.
- For each cover (bitmap) we need a separate matrix.

```
/* Start with the pre matrix */

/* Load identity matrix */
utMat4x4LoadIdentity(m_m44Pre);
```

```
/* Scale the dimension 0..2 to screen dimension */
utMat4x4Scale(m_m44Pre, GetWidth() / 2.0f, GetHeight() / 2.0f, 1);
/* Move the window from 0, 0 to 1, 1 coordinates */
utMat4x4Translate(m_m44Pre, 1.0f, 1.0f, 0);
/* An OpenGL like perspective calculation */
utMat4x4Perspective(m_m44Pre, s_fLensAngle, (float)GetWidth() / GetHeight(), (float)0.1, 100.0);
/* get a distance to the object */
utMat4x4Translate(m_m44Pre, 0, 0, s_fViewDist);

/* Now the cover movement */

/* Turn the view center point a little bit down */
utMat4x4RotX(m44, s_fViewAngle);
/* Move a little bit over the scene */
utMat4x4Translate(m44, 0, s_fViewPoint, 0);
/* Push the image on a circle */
utMat4x4Translate(m44, s_fCircleRadius * cos_angle, 0, s_fCircleRadius * sin_angle);
/* to get the 2-D lock we turn the cover here to correct the s_fViewAngle */
utMat4x4RotX(m44, -s_fViewAngle);
/* Scale it */
utMat4x4Scale(m44, s_fCoverScaling, s_fCoverScaling, 1.0f);

/* Now the post matrix */

/* Translate it to -1, -1. The center point is now 0 ,0 */
utMat4x4Translate(m_m44Post, -1.0f, -1.0f, 0);
/* Scale the cover bitmap of a size 0..2 */
utMat4x4Scale(m_m44Post, (float)2 / COVER_SIZE, (float)2 / COVER_SIZE, 1.0f);
```

As described all operations must be calculated for each frame for each cover. To reduce the effort it is spit into 3 parts. The pre and post matrix is constant over the scene. That's why it can be calculated once during the initialization. Only the dynamic part must be calculated for each cover. The final matrix is:

M = Mpre * Mdynamic * Mpost

For transformation of the 4∗4 to a 3∗2 matrix we use a utility function. The idea is just to remove the z component for the matrix. The z values are stored in the 3 line and 3 row.

Beside this the 4 line of the matrix must be removed. Such a conversion assumes that the 4x4 matrix realizes only affine transformations. This is only realized if the values are (0 0 1 Vscale). All matrix elements must be divided by Vscale.

```
utMat4x4ToMat3x2(mat44[0], pos.mat32);
```

6.10.2.3 z-order sorting

The hardware is not able to detect any z-order. The bitmaps will be drawn in the order as specified in the command list. Later drawn bitmaps are on top of previously drawn bitmaps. For this reason we calculate a z-value of the center of each bitmap manually by using the 4∗4 matrix.

```
utMat4x4GetXYZ(mat44[0], COVER_SIZE/2, COVER_SIZE/2, 0, &fX, &fY, &fZ);
```

To sort the draw order we just sort a list of all bitmaps:

```
qsort(&positions[0], (size_t)positions.size(), sizeof(positions[0]), CompareFnc);
```

The compare function is:

```
int Coverflow::CompareFnc( const void *arg1, const void *arg2 )
{
    Coverflow::COVERPOS *p1 = (Coverflow::COVERPOS *)arg1;
    Coverflow::COVERPOS *p2 = (Coverflow::COVERPOS *)arg2;

    if (p1->z < p2->z)
        return -1;
    if (p1->z > p2->z)
        return +1;
    return 0;
}
```

6.11 Tutorial: Digital Picture Frame

6.11.1 Summary

This example demonstrates several PixelEngine features in form of digital picture frame software.

It includes several blend classes to show different old picture, new picture animations by using different features like.

- Movements.
- Rotation.
- Alpha blend.
- Alpha masking.
- Color matrix modification.

6.12 Tutorial: Simple Drawing

6.12.1 Summary

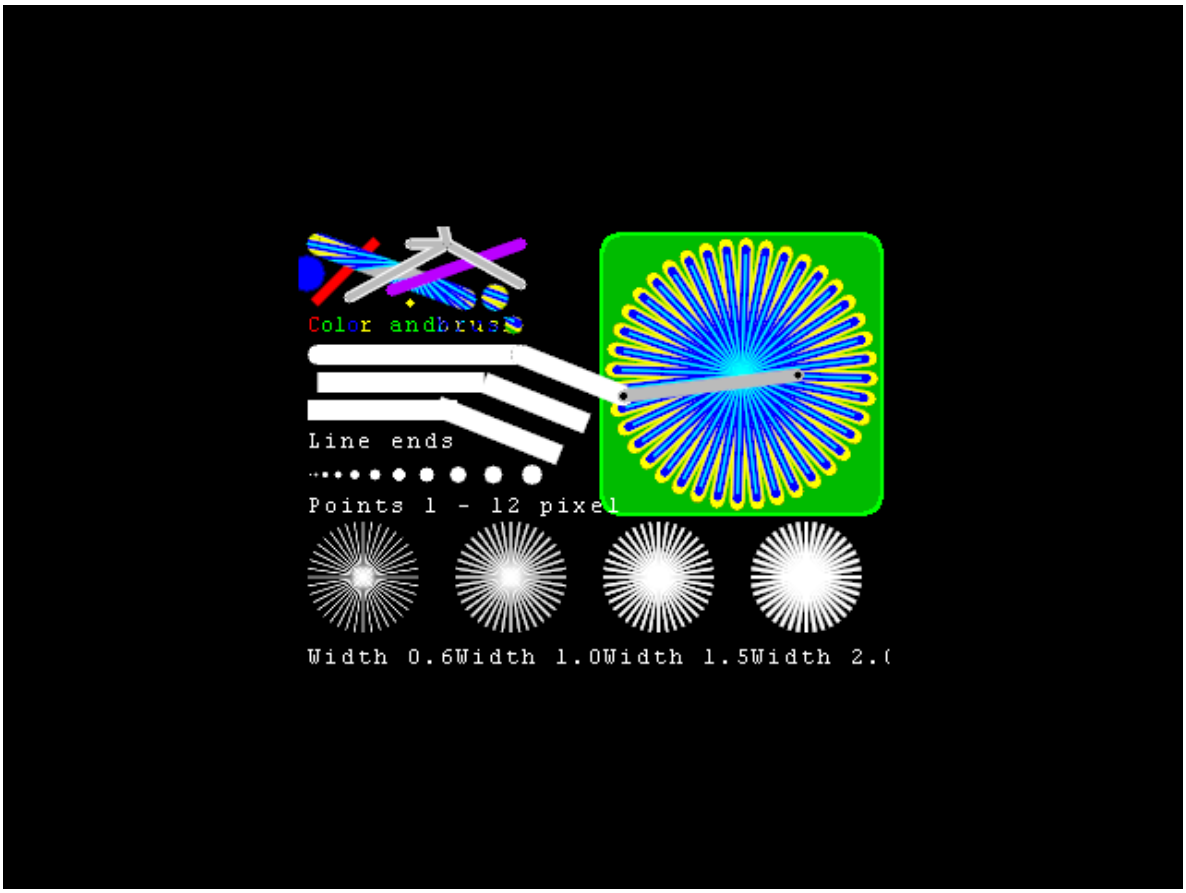
This is a more complex example that draws lines, circles, rectangles and text.

The example shows how complex features can be achieved in software by combining simple features supported by the hardware:

- Draw lines with different widths and line ends.

- Draw circles and points.
- Draw rectangles.

Figure 6-17 Expected result



Some information text is also displayed. The functions for drawing lines, circles and texts are implemented as utility functions in the util_lib directory.

6.12.2 Code documentation

6.12.2.1 Preparation

Once again, we start with a collection of several driver and display initializations.

```
/* Initialization of driver and display */
/* Initialize the driver */
UTIL_SUCCESS(ret,     mmlGdcSysInitializeDriver(0));
UTIL_SUCCESS(ret, utMmanReset() );

/* Allocate some of VRAM for Instruction buffer for the command sequencer. Note, that mmlGdcVideoAlloc is an application defined routine to manage the VRAM space. The 2D core driver does not include any management of the VRAM. */
vInstrBuffer   =   mmlGdcVideoAlloc(fifo_size,   0,   NULL);
UTIL_SUCCESS(ret,     mmlGdcSysSetInstructionBuffer(vInstrBuffer, fifo_size));
```

We need target surface in VRAM to render strings in it.

```
/* Allocate our buffers */
for (i = 0; i < BUFFER_COUNT; i++) {
    UTIL_SUCCESS(ret, mmlGdcSmResetSurfaceObject(&target_c[i]));
    target[i] = &target_c[i];
    UTIL_SUCCESS(ret, utSurfCreateBuffer(target[i], windowProp.width, windowProp.height,
MML_GDC_SURF_FORMAT_R5G6B5));
```

We want to print some information on the screen. We can set up a font by using the utility functions of this tutorial with the code:

```
/* Load a font bitmap */
mmlGdcSmGenSurfaceObjects(1, &sFont);
UTIL_SUCCESS(ret, utSurfLoadBitmap(sFont, courier_12, MM_FALSE));

/* Set it as font for the util lib */
utSetFont(&utCtx, sFont);
```

Then we create a point 50 pixels in diameter (actually a bitmap) again using utility functions.

```
/* Initialize the driver */
UTIL_SUCCESS(ret, mmlGdcSysInitializeDriver(0));
UTIL_SUCCESS(ret, utMmanReset() );

/* Allocate some of VRAM for Instruction buffer for the command sequencer. Note, that mmlGdcVideoAlloc is
an application defined routine to manage the VRAM space. The 2D core driver does not include any
management of the VRAM. */
vInstrBuffer = mmlGdcVideoAlloc(fifo_size, 0, NULL);
UTIL_SUCCESS(ret, mmlGdcSysSetInstructionBuffer(vInstrBuffer, fifo_size));

/* Setup and enable the display */
UTIL_SUCCESS(ret, mmlGdcDispOpenDisplay(&dispParams, &display));

windowProp.topLeftX = (dispParams.xResolution - BGR_WIDTH) / 2;
windowProp.topLeftY = (dispParams.yResolution - BGR_HEIGHT) / 2;
windowProp.width = BGR_WIDTH;
windowProp.height = BGR_HEIGHT;
windowProp.layerId = MML_GDC_DISP_LAYER_0;

/* Create a window and assign it as layer 0 */
```

```
UTIL_SUCCESS(ret, mmlGdcDispWinCreate(display, &windowProp, &win));

/* Allocate our buffers */
for (i = 0; i < BUFFER_COUNT; i++) {
    UTIL_SUCCESS(ret, mmlGdcSmResetSurfaceObject(&target_c[i]));
    target[i] = &target_c[i];
    UTIL_SUCCESS(ret, utSurfCreateBuffer(target[i], windowProp.width, windowProp.height,
MML_GDC_SURF_FORMAT_R5G6B5));
}

utResetContext(&utCtx);

/* Load a font bitmap */
mmlGdcSmGenSurfaceObjects(1, &sFont);
UTIL_SUCCESS(ret, utSurfLoadBitmap(sFont, courier_12, MM_FALSE));

/* Set it as font for the util lib */
utSetFont(&utCtx, sFont);

/* Initialize a point bitmap */
UTIL_SUCCESS(ret, utInitPoint(&utCtx, 50));
```

6.12.2.2 Doing the animation

Now begin with the animation: slightly different scenes 360 times. First clear the screen.

```
/* Clear the whole frame (You should optimize and only redraw the changed parts) */
UTIL_SUCCESS(ret, utRect(&utCtx, 0, 0, 320, 240));
```

Then draw a rounded rectangle with a border.

```
/* Now we draw a RoundRect with border. The simplest way is to draw it twice with different sizes and color
*/
utColor(&utCtx, 0, 255, 0, 255);
UTIL_SUCCESS(ret, utRoundRect(&utCtx, 163, 83, 154, 154, 10, 10));
```

And then draw the different parts of the complete scene.

```
/* The Drawing part is split into 5 sections. The result of DrawSample is used for DrawMix. Don't change the
order!   */
UTIL_SUCCESS(ret,   DrawLines(&utCtx));
UTIL_SUCCESS(ret, utInitPoint(&utCtx, 20));
```

```
UTIL_SUCCESS(ret,     DrawPoints(&utCtx));
UTIL_SUCCESS(ret, DrawSample(&utCtx, f));
UTIL_SUCCESS(ret, DrawLineEnds(&utCtx, f));
UTIL_SUCCESS(ret, DrawMix(&utCtx, target[nCurBuffer]));
```

6.12.3 The drawing functions

6.12.3.1 Drawlines

We draw some "flowers" made of lines at different angles centered on the same point. First set up the line width and line end, write some information and then draw the flower.

```
/* Draw 4 "flowers" with lines of different width */
MM_U32    DrawLines(UTIL_CONTEXT    *putCtx)
{
    MM_U32 ret = MML_OK;
    MM_FLOAT f, cx, cy, px, py, s, c;

    /* Set the line end to round */
    utLineEnd(putCtx, UT_LINE_END_BUTT);

    px = 5;
    py = 0;

    /* The the paint color */
    utColor(putCtx,    255,    255,    255,    255);
    /* Change the line width to 0.5 pixel */
    utLineWidth(putCtx, 0.6f);
    /* Print a text as comment */
    UTIL_SUCCESS(ret, utTextOut(putCtx, (MM_S32)px, (MM_S32)py, "Width 0.6"));
    cx = 30 + px;
    cy = 50 + py;
    /* Draw lines in a loop */
    for (f = 0; f < 180; f+= 9.0f)
    {
        c = 30 * cosf(f*DegreeToPI);
        s = 30 * sinf(f*DegreeToPI);
        UTIL_SUCCESS(ret, utLinef(putCtx, cx - c, cy - s, cx + c, cy + s));
    }
```

Repeat with different line widths at different positions on the screen.

```
/* repeat the code with different offsets and line width */
```

```
px += 80;

utLineWidth(putCtx, 1.0f);
UTIL_SUCCESS(ret, utTextOut(putCtx, (MM_S32)px, (MM_S32)py, "Width 1.0"));
cx = 30 + px;
cy = 50 + py;
for (f = 0; f < 180; f+= 9.0f)
{
    c = 30 * cosf(f*DegreeToPI);
    s = 30 * sinf(f*DegreeToPI);
    UTIL_SUCCESS(ret, utLinef(putCtx, cx - c, cy - s, cx + c, cy + s));
}

px += 80;

utLineWidth(putCtx, 1.5f);
UTIL_SUCCESS(ret, utTextOut(putCtx, (MM_S32)px, (MM_S32)py, "Width 1.5"));
cx = 30 + px;
cy = 50 + py;
for (f = 0; f < 180; f+= 9.0f)
{
    c = 30 * cosf(f*DegreeToPI);
    s = 30 * sinf(f*DegreeToPI);
    UTIL_SUCCESS(ret, utLinef(putCtx, cx - c, cy - s, cx + c, cy + s));
}

px += 80;

utLineWidth(putCtx, 2.0f);
UTIL_SUCCESS(ret, utTextOut(putCtx, (MM_S32)px, (MM_S32)py, "Width 2.0"));
cx = 30 + px;
cy = 50 + py;
for (f = 0; f < 180; f+= 9.0f)
{
    c = 30 * cosf(f*DegreeToPI);
    s = 30 * sinf(f*DegreeToPI);
    UTIL_SUCCESS(ret, utLinef(putCtx, cx - c, cy - s, cx + c, cy + s));
}
return ret;
```

6.12.3.2 Drawpoints

Draws 12 points with different sizes using the utility function.

```
/* Now paint */
for (i = 1; i < 12; i++)
{
    utPointSize(putCtx, (MM_FLOAT)i);
    UTIL_SUCCESS(ret, utPoint(putCtx, 5 + i * i, 105));
}
```

6.12.3.3 DrawSample

This function draws a flower with different line widths and colors.

6.12.3.4 DrawLineEnds

Draws the end of the lines by combining line and point draw utility functions.

6.12.3.5 DrawMix

Draws some text, lines and points with different colors or a brush.

7. Module Index

7.1 Modules

Here is a list of all modules:

- Basic Graphics
  - Driver Initialization API
  - Configuration API
  - Surface API
  - Display API
  - Pixel Engine API
  - Synchronization API
  - 2D Core Interrupt Controller API
  - Error Reporting API
  - Error Codes
  - Basic Graphics Type Definitions
  - Version Numbers
- Type Definition
- Macro Definition
- Tutorial Utility Library
  - Utilities for the Memory Management
  - Utility functions for matrix calculations
  - Utilities for the compatibility with other drivers
  - Utilities for the Surface Management
  - Utilities for the compression
    - Utilities for RLA (run length adaptive compression)
    - Utilities for RLC (run length compression)
  - Util class collection
    - CCtx
    - CDevice
    - CDisplay
    - CMenu
    - CSurface
    - CWindow

8. Hierarchical Index

8.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

RLAD::BitStream

CCtx

CDevice

CDisplay

CMenuItem

CSurface< NUM_BUFFERS >

CSurface< 1 >

CTextWindow

  CMenu

CWindow

  CSurfaceWindow< NUM_BUFFERS >

  CSurfaceWindow< 1 >

    CStaticSurfaceWindow

RLAD::Frame

MML_GDC_DISP_MODE_LINE

MML_GDC_DISP_PROPERTIES

MML_GDC_DISP_TCON_PROPERTIES

MML_GDC_DISP_WINDOW_PROPERTIES

MML_GDC_PE_CONTEXT_CONTAINER

MML_GDC_SURFACE_CONTAINER

MML_GDC_SYNC_CONTAINER

MML_GDC_SYSINIT_INFO

RLAD::Package

RLAD::Frame::Pixel

RLAD

9. Data Structure Index

9.1 Data Structures

Here are the data structures with brief descriptions:

RLAD::BitStream

CCtx

CDevice

CDisplay

CMenu

CMenuItem

CStaticSurfaceWindow

CSurface< NUM_BUFFERS >

CSurfaceWindow< NUM_BUFFERS >

CWindow

RLAD::Frame

MML_GDC_DISP_MODE_LINE

MML_GDC_DISP_PROPERTIES

MML_GDC_DISP_TCON_PROPERTIES

MML_GDC_DISP_WINDOW_PROPERTIES

MML_GDC_PE_CONTEXT_CONTAINER

MML_GDC_SURFACE_CONTAINER

MML_GDC_SYNC_CONTAINER

MML_GDC_SYSINIT_INFO

RLAD::Package

RLAD::Frame::Pixel

RLAD

10. File Index

10.1 File List

Here is a list of all documented files with brief descriptions:

flash_resource.h
: Include this file before the definition of a bitmap

mm_defines.h
: Common macro definitions for all modules

mm_gdc_erp.h
: Error Reporting API

mm_gdc_errors.h
: Error Codes for the Basic Graphics modules

mm_gdc_module_id.h
: Basic Graphics module ids (common)

mm_gdc_version.h
: Basic Graphics Driver Version Numbers

mm_types.h
: Basic type definitions

mmd_gdc_interrupthandler.h
: 2D Core Interrupt Controller API

mml_gdc_config.h
: Controls global graphics driver and hardware configurations

mml_gdc_display.h
: Display API

mml_gdc_erp.h
: Error Reporting API

mml_gdc_pixeng.h
: Pixel Engine API

mml_gdc_surfman.h
: Surface Manager Interface

mml_gdc_sync.h
: Synchronization of framebuffer operations

mml_gdc_sysinit.h
: Driver Initialization Module

pe_matrix.h
: Provide some matrix utility functions

sm_util.h
: This is just a helper implementation for development and will be removed in the final version

ut_class_ctx.h
: This class abstracts an MML_GDC_PE_CONTEXT

ut_class_device.h

This class abstracts the device initialization

ut_class_display.h

This class abstracts the display initialisation

ut_class_menu.h

This class realizes a simple menu

ut_class_rlad.h

This sample code can be used to compress a buffer using the MML_GDC_SURF_COMP_RLA, MML_GDC_SURF_COMP_RLAD or ::MML_GDC_SURF_COMP_RLAD_UNIFORM format

ut_class_surface.h

This class abstracts MML_GDC_SURFACE objects

ut_class_window.h

This class abstracts windows

ut_compatibility.h

This file defines some interfaces that are part of other drivers. The util library implements very simple instances of it but they must be not used for software products. However it allows to run the sample applications

ut_compression.h

This file defines a helper function that can be used to compress a surface

ut_memman.h

This file defines some interfaces for the memory management

ut_rlc.h

This sample code can be used to create a run-length encoded buffer

11. Module Documentation

11.1 Basic Graphics

This section collects all APIs of the driver.

Modules

- Driver Initialization API

  The Driver Initialization API exposes functions to initialize and uninitialize the driver.

- Configuration API

  The Configuration API allows changing or reading global graphics driver configurations or status information.

- Surface API

  The Surface API provides all functions to manage memory blocks with image content, called image buffer. (See also Surface Overview)

- Display API

  The Display API exposes all the hardware features of the display unit. See also Display Overview.

- Pixel Engine API

  Pixel Engine (PixEng) API.

- Synchronization API

  Synchronization API - Synchronization of framebuffer operations.

- 2D Core Interrupt Controller API

  2D Core Interrupt Controller handler functions

- Error Reporting API

  Error Reporting API - Error Reporting for selected modules and level.

- Error Codes

  Error Codes of this driver.

- Basic Graphics Type Definitions
- Version Numbers

  The Version numbers of this driver.

11.1.1 Detailed Description

This section collects all APIs of the driver. The collection of APIs includes:

- APIs for the access of the hardware units (e.g. Displays, Pixel Engine (2D Rendering)).
- APIs for services like driver initialization, synchronization, surface management, configuration.

11.2 Driver Initialization API

The Driver Initialization API exposes functions to initialize and uninitialize the driver.

Data Structures

- struct MML_GDC_SYSINIT_INFO

Macros

- #define GFX_PLL_MIN 20000000U
- #define GFX_PLL_MAX 415000000U

Functions

- MM_ERROR mmlGdcSysInitializeDriver (MML_GDC_SYSINIT_INFO *pDriverInitInfo)
- MM_ERROR mmlGdcSysUninitializeDriver (void)
- MM_ERROR mmlGdcSysSetInstructionBuffer (void *address, MM_U32 size)

Default initializer

- #define MML_GDC_SYSINIT_INITIALIZER

Resource names

- #define MM_GDC_RES_DISP0 (1U << 0U)
- #define MM_GDC_RES_LAYER0 (1U << 1U)
- #define MM_GDC_RES_LAYER1 (1U << 2U)
- #define MM_GDC_RES_FETCH_DECODE0 (1U << 3U)
- #define MM_GDC_RES_FETCH_LAYER0 (1U << 4U)

11.2.1 Detailed Description

The Driver Initialization API exposes functions to initialize and uninitialize the driver.

#include "mml_gdc_sysinit.h"

11.2.2 Macro Definition Documentation

11.2.2.1 #define GFX_PLL_MAX 415000000U

maximum GFX PLL 415 MHz

11.2.2.2 #define GFX_PLL_MIN 20000000U

Allowed PLL frequency range minimum GFX PLL 20 MHz

11.2.2.3 #define MML_GDC_SYSINIT_INITIALIZER

Value:

```
{     ¥
    0U,             /* no safety driver */   ¥
    320000000U   /* GFX PLL 320 MHz     */ ¥
}
```

11.2.3 Function Documentation

11.2.3.1 MM_ERROR mmlGdcSysInitializeDriver(MML_GDC_SYSINIT_INFO *pDriverInitInfo)

Used to initialize the driver at startup. Applications must initialize the driver before they can call other driver functions.

Note:

- *The 2D core hardware must be in default state, i.e. no registers may be altered between HW reset and the call of mmlGdcSysInitializeDriver(). The only except exception are the LockUnlock registers, which can be used by a safety driver to protect specific streams against non-privileged access. The registers related to these streams may also be altered by the safety driver before mmlGdcSysInitializeDriver() is called.*

Parameters

pDriverInitInfo	Can be NULL or a pointer to a MML_GDC_SYSINIT_INFO driver initialization structure.

Return values

MML_OK	Successfully initialized driver
MML_ERR_GDC_SYS_DEVICE_INVALID_PARAMETER	GfxPll parameter out of range.
MML_ERR_GDC_SYS_DEVICE_ALREADY_INITIALIZED	already initialized.
MML_ERR_GDC_SYS_DEVICE_INIT_FAILED	Initialization of the driver failed.

11.2.3.2 MM_ERROR mmlGdcSysSetInstructionBuffer(void *address, MM_U32 size)

Assign internal VRAM for command queue

Note:

- *The command queue is required to buffer the render instructions to allow a non-blocking API handling. The required instruction buffer size depends on the amount and complexity of the render instructions and which synchronization instructions are used. A recommended size is 8 kByte. To get information about the instruction buffer usage an application can use the mmlGdcConfigGetAttribute function with attribute MML_GDC_CONFIG_ATTR_MIN_INSTRUCTION_BUFFER. The function must be called after mmlGdcSysInitializeDriver before any render or display operations. A reconfiguration of the instruction buffer is not possible.*

Parameters

address	Start address in the VRAM. Must be 32 byte aligned.
size	Size in bytes of the buffer to assign, must be DWORD (4 Bytes) aligned, the function will return an error otherwise. The application must ensure that the address as well as the address plus size are within the range of the 2D core VRAM memory, the function will not do this.

Return values

MML_OK	on success, otherwise the related error code

11.2.3.3 MM_ERROR mmlGdcSysUninitializeDriver(void)

Used to shutdown the driver. Applications must uninitialize the driver after calling mmlGdcSysInitializeDriver.

Return values

MML_OK	Successfully shutdown the driver
MML_ERR_GDC_SYS_DEVICE_NOT_YET_INITIALIZED	not yet initialized.
MML_ERR_GDC_SYS_DEVICE_CLOSE_FAILED	Driver shutdown failed.

11.3 Configuration API

The Configuration API allows changing or reading global graphics driver configurations or status information.

Enumerations

- enum MML_GDC_CONFIG_ATTR {
  MML_GDC_CONFIG_ATTR_MAJOR_VERSION = 0,
  MML_GDC_CONFIG_ATTR_MINOR_VERSION,
  MML_GDC_CONFIG_ATTR_BUILD_VERSION,
  MML_GDC_CONFIG_ATTR_MIN_INSTRUCTION_BUFFER,
  MML_GDC_CONFIG_ATTR_CURRENT_INSTRUCTION_BUFFER,
  MML_GDC_CONFIG_ATTR_DISPLAY_NOBLOCK,
  MML_GDC_CONFIG_ATTR_BUILD_TYPE
  }

Functions

- MM_ERROR mmlGdcConfigSetAttribute (MML_GDC_CONFIG_ATTR pname, MM_U32 param)
- MM_ERROR mmlGdcConfigGetAttribute (MML_GDC_CONFIG_ATTR pname, MM_U32 *pParam)

Detailed Description

The Configuration API allows changing or reading global graphics driver configurations or status information.

#include "mml_gdc_config.h"

11.3.1 Enumeration Type Documentation

11.3.1.1 enum MML_GDC_CONFIG_ATTR

Enumeration of the different config attributes

Enumerator

MML_GDC_CONFIG_ATTR_MAJOR_VERSION
Returns the major version of the driver. This is a read only attribute. Setting this attribute will result in error.

MML_GDC_CONFIG_ATTR_MINOR_VERSION
Returns the minor version of the driver. This is a read only attribute. Setting this attribute will result in error.

MML_GDC_CONFIG_ATTR_BUILD_VERSION

Returns the build version of the driver. This is a read only attribute. Setting this attribute will result in error.

MML_GDC_CONFIG_ATTR_MIN_INSTRUCTION_BUFFER

This attribute is only available in mmlGdcConfigGetAttribute(). The returned value represents the smallest available InstructionBuffer in bytes during all calls. A function call with this parameter resets the measurement. The returned value can be used by an application to measure the usage of the instruction buffer assigned by mmlGdcSysSetInstructionBuffer.

MML_GDC_CONFIG_ATTR_CURRENT_INSTRUCTION_BUFFER

This attribute is only available in mmlGdcConfigGetAttribute(). The returned value represents the current available InstructionBuffer in bytes. The returned value can be used by an application to decide whether or not further render steps should be delayed and continued later because the hardware is currently still busy.

MML_GDC_CONFIG_ATTR_DISPLAY_NOBLOCK

The following functions can not be executed, if a previous reconfiguration of a corresponding window or display is not yet finished:

- mmlGdcDispOpenDisplay
- mmlGdcDispCloseDisplay
- mmlGdcDispWinCreate
- mmlGdcDispWinDestroy
- mmlGdcDispCommit
- mmlGdcDispWinCommit

If this attribute is 0 (default), the function will block the CPU until it can be executed. If this attribute is not 0, the function will return immediately in that case with error MML_ERR_GDC_DISP_DEV_BUSY. It is up to the application to handle this case and reschedule the function call later.

Note:

- *The application can also use the synchronization API to find out if the previous reconfiguration is finished.*

MML_GDC_CONFIG_ATTR_BUILD_TYPE

Returns the build type of the driver. The returned values can be ’d’ for debug version of driver ’r’ for release version of driver ’p’ for production version of driver This is a read only attribute. Setting this attribute will result in error.

11.3.2 Function Documentation

11.3.2.1 MM_ERROR mmlGdcConfigGetAttribute(MML_GDC_CONFIG_ATTR pname, MM_U32 *pParam)

Gets the current value of a graphics library attribute. See MML_GDC_CONFIG_ATTR for a list of attributes.

Parameters

in	pname	Name of the attribute to get. See MML_GDC_CONFIG_ATTR
out	pParam	Address where the read value of the attribute is stored

Return values

MML_OK	on success
MML_ERR_GDC_CONFIG_INVALID_PARAMETER	if pname is invalid
MML_ERR_GDC_CONFIG_INTERNAL_ERROR	if value could not be retrieved

11.3.2.2 MM_ERROR mmlGdcConfigSetAttribute(MML_GDC_CONFIG_ATTR pname, MM_U32 param)

Sets a graphics library attribute. See MML_GDC_CONFIG_ATTR for a list of attributes.

Parameters

in	pname	Name of the attribute to set. See MML_GDC_CONFIG_ATTR
in	param	Value to set for the attribute

Return values

MML_OK	on success
MML_ERR_GDC_CONFIG_INVALID_PARAMETER	if a parameter is invalid
MML_ERR_GDC_CONFIG_INTERNAL_ERROR	if value could not be set

11.4 Surface API

The Surface API provides all functions to manage memory blocks with image content, called image buffer. (See also Surface Overview)

Data Structures

- struct MML_GDC_SURFACE_CONTAINER

Macros

- #define MML_GDC_SURFACE_MAX_WIDTH 4096
- #define MML_GDC_SURFACE_MAX_HEIGHT 4096
- #define MML_GDC_SURFACE_CONTROL_WIDTH 2048
- #define MML_GDC_SURFACE_CONTROL_HEIGHT 2048

Typedefs

- typedef MML_GDC_SURFACE_CONTAINER * MML_GDC_SURFACE

Enumerations

- enum MML_GDC_SURF_FORMAT {
  MML_GDC_SURF_FORMAT_R8G8B8A8 = 0x00,
  MML_GDC_SURF_FORMAT_A8B8G8R8,
  MML_GDC_SURF_FORMAT_A8R8G8B8,
  MML_GDC_SURF_FORMAT_B8G8R8A8,
  MML_GDC_SURF_FORMAT_R8G8B8X8,
  MML_GDC_SURF_FORMAT_X8B8G8R8,

```
        MML_GDC_SURF_FORMAT_X8R8G8B8,
        MML_GDC_SURF_FORMAT_R8G8B8,
        MML_GDC_SURF_FORMAT_B8G8R8,
        MML_GDC_SURF_FORMAT_R6G6B6,
        MML_GDC_SURF_FORMAT_R4G4B4A4,
        MML_GDC_SURF_FORMAT_A4R4G4B4,
        MML_GDC_SURF_FORMAT_R5G5B5A1,
        MML_GDC_SURF_FORMAT_A1R5G5B5,
        MML_GDC_SURF_FORMAT_A1B5G5R5,
        MML_GDC_SURF_FORMAT_B5G5R5A1,
        MML_GDC_SURF_FORMAT_R5G6B5,
        MML_GDC_SURF_FORMAT_A8RGB8,
        MML_GDC_SURF_FORMAT_RGB8,
        MML_GDC_SURF_FORMAT_A8,
        MML_GDC_SURF_FORMAT_A4RGB4,
        MML_GDC_SURF_FORMAT_A4,
        MML_GDC_SURF_FORMAT_A2,
        MML_GDC_SURF_FORMAT_A1,
        MML_GDC_SURF_FORMAT_RGB1
    }
- enum MML_GDC_SURF_COMP {
        MML_GDC_SURF_COMP_NON = 0x4,
        MML_GDC_SURF_COMP_RLC = 0x3,
        MML_GDC_SURF_COMP_RLA = 0x2,
        MML_GDC_SURF_COMP_RLAD = 0x0
    }
- enum MML_GDC_SURF_CLF {
        MML_GDC_SURF_CLF_R8G8B8,
        MML_GDC_SURF_CLF_B8G8R8,
        MML_GDC_SURF_CLF_R5G5B5,
        MML_GDC_SURF_CLF_A1R5G5B5,
        MML_GDC_SURF_CLF_A4R4G4B4
    }
- enum MML_GDC_SURF_CLM {
        MML_GDC_SURF_CLM_NEUTRAL = 0x0,
        MML_GDC_SURF_CLM_INDEX_RGB,
        MML_GDC_SURF_CLM_INDEX_RGBA
    }
- enum MML_GDC_SURF_ATTR {
        MML_GDC_SURF_ATTR_BASE_ADDRESS = 0x0,
        MML_GDC_SURF_ATTR_PHYS_ADDRESS,
        MML_GDC_SURF_ATTR_BASE_ADDRESS2,
        MML_GDC_SURF_ATTR_PHYS_ADDRESS2,
        MML_GDC_SURF_ATTR_WIDTH,
        MML_GDC_SURF_ATTR_HEIGHT,
        MML_GDC_SURF_ATTR_STRIDE,
        MML_GDC_SURF_ATTR_BITPERPIXEL,
        MML_GDC_SURF_ATTR_COLORBITS,
        MML_GDC_SURF_ATTR_COLORSHIFT,
        MML_GDC_SURF_ATTR_COMPRESSION_FORMAT,
        MML_GDC_SURF_ATTR_RLAD_MAXCOLORBITS,
        MML_GDC_SURF_ATTR_SIZEINBYTES,
        MML_GDC_SURF_ATTR_CLUTMODE,
```

```
    MML_GDC_SURF_ATTR_CLUTCOUNT,
    MML_GDC_SURF_ATTR_CLUTBITPERPIXEL,
    MML_GDC_SURF_ATTR_CLUTCOLORBITS,
    MML_GDC_SURF_ATTR_CLUTCOLORSHIFT,
    MML_GDC_SURF_ATTR_CLUTBUFFERADDRESS,
    MML_GDC_SURF_ATTR_CLUTBUFFER_PHYS_ADDRESS,
    MML_GDC_SURF_ATTR_SURF_FORMAT,
    MML_GDC_SURF_ATTR_USERDEFINED
}
```

Functions

- MM_ERROR mmlGdcSmResetSurfaceObject(MML_GDC_SURFACE surf)
- MM_ERROR mmlGdcSmAssignBuffer(MML_GDC_SURFACE surf, MM_U32 uWidth, MM_U32 uHeight, MML_GDC_SURF_FORMAT eFormat, void *pBufferAddress, MM_U32 uRleWords)
- MM_ERROR mmlGdcSmAssignClut(MML_GDC_SURFACE surf, MML_GDC_SURF_CLM eMode, MM_U32 uCount, MML_GDC_SURF_CLF eFormat, void *pBufferAddress)
- MM_ERROR mmlGdcSmSetAttribute(const MML_GDC_SURFACE surf, MML_GDC_SURF_ATTR eName, MM_U32 uValue)
- MM_ERROR mmlGdcSmGetAttribute(const MML_GDC_SURFACE surf, MML_GDC_SURF_ATTR eName, MM_U32 *puValue)

11.4.1 Detailed Description

The Surface API provides all functions to manage memory blocks with image content, called image buffer. (See also Surface Overview)

```
#include "mml_gdc_surfman.h"
```

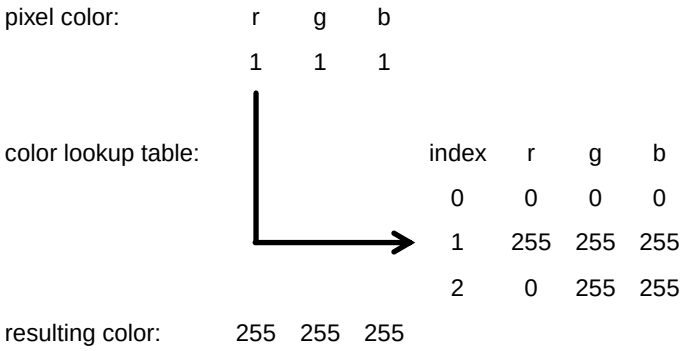
The Surface API provides all functions to manage memory blocks with image content, called image buffer. A "surface" is a description of such an image buffer, including dimension of the image, pixel format and physical address in memory. The described image can be a (compressed) RGB(A) buffer and optionally use a color lookup table.

Most modules of this driver can work on surfaces. Examples are display and PixEng. The Surface API allows for instance to:

- Pass surfaces created by the application to PixEng for further processing
- Pass surfaces created by the application to Display for displaying on the screen, etc.

  The properties can be assigned by using the helper functions mmlGdcSmAssignBuffer and mmlGdcSmAssignClut or "manually" using mmlGdcSmSetAttribute calls. In the second case the minimal required attributes are typically MML_GDC_SURF_ATTR_BASE_ADDRESS, MML_GDC_SURF_ATTR_WIDTH, MML_GDC_SURF_ATTR_HEIGHT, MML_GDC_SURF_ATTR_BITPERPIXEL, MML_GDC_SURF_ATTR_COLORBITS and MML_GDC_SURF_ATTR_COLORSHIFT.

Optionally a color lookup table can be defined for the image. A color lookup table is a list with a defined number of red, green, blue and optionally alpha values. If an index table is defined for an image the blue and green color parts, if the image are not longer used but the red component is used as "pointer" to the color lookup table and the related red, green, blue (and alpha) value define the pixel color.



A color lookup table can be defined with the helper function mmlGdcSmAssignClut or "manually" using mmlGdcSmSetAttribute calls. In the second case the minimal required attributes are MML_GDC_SURF_ATTR_CLUTMODE, MML_GDC_SURF_ATTR_CLUTCOUNT, MML_GDC_SURF_ATTR_CLUTBITPERPIXEL, MML_GDC_SURF_ATTR_CLUTCOLORBITS, MML_GDC_SURF_ATTR_CLUTCOLORSHIFT and MML_GDC_SURF_ATTR_CLUTBUFFERADDRESS.

Note:

- *The maximal bits per pixel for an index entry is 24. Therefore if an alpha channel is required a reduced bit with for the color channels are required (e.g., R6G6B6A6). If this is not enough the alpha channel can also be stored in the image pixel.*
  If an indexed image is used it is required that the red component starts at the lowest bit for each pixel (see MML_GDC_SURF_ATTR_COLORBITS).

11.4.2 Macro Definition Documentation

11.4.2.1 #define MML_GDC_SURFACE_CONTROL_HEIGHT 2048

Maximum supported height for surfaces .

11.4.2.2 #define MML_GDC_SURFACE_CONTROL_WIDTH 2048

Maximum supported width for surfaces.

11.4.2.3 #define MML_GDC_SURFACE_MAX_HEIGHT 4096

Absolute maximum height for surfaces.

11.4.2.4 #define MML_GDC_SURFACE_MAX_WIDTH 4096

Absolute maximum width for surfaces.

11.4.3 Typedef Documentation

11.4.3.1 typedef MML_GDC_SURFACE_CONTAINER* MML_GDC_SURFACE

The surface object definition

11.4.4 Enumeration Type Documentation

11.4.4.1 enum MML_GDC_SURF_ATTR

Surface attribute.

Enumerator

MML_GDC_SURF_ATTR_BASE_ADDRESS
Virtual base address (initial: 0).

Note:
- *The base address should be used to address images inside the VRAM. While setting a virtual address the physical address will be overwritten.*

MML_GDC_SURF_ATTR_PHYS_ADDRESS
Physical base address (initial: 0).

Note:
- *Can be used to read image buffers direct from NOR flash. While setting a physical address the virtual address will be overwritten.*

MML_GDC_SURF_ATTR_BASE_ADDRESS2
Not used for S6E2D! Virtual base address of the UV buffer (initial: 0).

MML_GDC_SURF_ATTR_PHYS_ADDRESS2
Not used for S6E2D! Physical base address of the UV buffer (initial: 0).

MML_GDC_SURF_ATTR_WIDTH
Width in pixels (initial: 0). MML_GDC_SURF_ATTR_HEIGHT Height in pixels (initial: 0). MML_GDC_SURF_ATTR_STRIDE Size of a line in bytes (initial: 0).

Note:
- *If stride is 0, the default stride for the image buffer is assumed represented by the following formula:*
  *stride = ((Width * BitPerPixel + 7) >> 3)*
  The GetAttribute call will return the previously set "custom" stride value or the default stride calculated with the formula above. The stride value is not important for compressed images. If the surface describes a compressed image the returned value will be 0.

MML_GDC_SURF_ATTR_BITPERPIXEL
Size of one pixel in bits. Can be one of 1, 2, 4, 8, 12, 16, 24, 32(initial: 32).

MML_GDC_SURF_ATTR_COLORBITS
Color component size in bits 0xRRGGBBAA or 0xY0U0Y1V0 (initial: 0x08080808).
color_bits = red_bits<<24 + green_bits<<16 + blue_bits<<8 + alpha_bits //for RGBA format,

MML_GDC_SURF_ATTR_COLORSHIFT
Color component shift (0xRRGGBBAA) or (0xY0U0Y1V0) (initial: 0x18100800).
color_shift = red_shift<<24 + green_shift<<16 + blue_shift<<8 + alpha_shift //for RGBA format,

MML_GDC_SURF_ATTR_COMPRESSION_FORMAT
Compression format (must be one of MML_GDC_SURF_COMP, initial MML_GDC_SURF_COMP_NON).

MML_GDC_SURF_ATTR_RLAD_MAXCOLORBITS
Maximum for average number of bits per compressed pixel. This value is used for surfaces with compression format MML_GDC_SURF_COMP_RLAD. The format is analog to MML_GDC_SURF_ATTR_COLORBITS (0xRRGGBBAA) or (0xY0U0Y1V0). The initial value is 0x08080808. If the surface is used as target buffer (blit) and the compression format is MML_GDC_SURF_COMP_RLAD, the RLAD_BITPERPIXEL value defines the maximum write buffer size (see MML_GDC_SURF_ATTR_SIZEINBYTES). The application can use the

MML_GDC_SURF_ATTR_SIZEINBYTES parameter to calculate the required buffer size and can allocate and assign a VRAM space for this operation.

MML_GDC_SURF_ATTR_SIZEINBYTES

Buffer size in bytes (initial: 0).

Note:

- *This value must be set for images with compression type MML_GDC_SURF_COMP_RLC and MML_GDC_SURF_COMP_RLA. The size can be set to zero for all other image types. If size is zero mmlGdcSmGetAttribute will return the following size depending of the given compression type:*

- MML_GDC_SURF_COMP_NON: required buffer size (Height ∗ Stride).
- MML_GDC_SURF_COMP_RLC: 0 (correct size must be set by application).
- MML_GDC_SURF_COMP_RLA: 0 (correct size must be set by application).
- MML_GDC_SURF_COMP_RLAD: the maximal required size for the given compression settings.

MML_GDC_SURF_ATTR_CLUTMODE

Color look up table mode (must be one of MML_GDC_SURF_CLM, initial MML_GDC_SURF_CLM_NEUTRAL).

MML_GDC_SURF_ATTR_CLUTCOUNT

Number of color look up table entries (0..255, initial: 0 = no color look up table).

MML_GDC_SURF_ATTR_CLUTBITPERPIXEL

Size of one entry in bits (1, 2, 4, 8, 16, 24, 32, initial: 0).

MML_GDC_SURF_ATTR_CLUTCOLORBITS

Color component size of one entry in bits (0xRRGGBBAA), initial: 0.

MML_GDC_SURF_ATTR_CLUTCOLORSHIFT

Color component shift of one entry in bits (0xRRGGBBAA), initial: 0.

MML_GDC_SURF_ATTR_CLUTBUFFERADDRESS

Virtual address of CLUT data, initial: 0.

ML_GDC_SURF_ATTR_CLUTBUFFER_PHYS_ADDRESS

Physical address of CLUT data, initial:0.

MML_GDC_SURF_ATTR_SURF_FORMAT

Macro attribute to set and get MML_GDC_SURF_FORMAT

Note:

- *A mmlGdcSmSetAttribute call with the attribute MML_GDC_SURF_ATTR_SURF_FORMAT will implicitly set the attributes MML_GDC_SURF_ATTR_BITPERPIXEL, MML_GDC_SURF_ATTR_COLORBITS and MML_GDC_SURF_ATTR_COLORSHIFT. A mmlGdcSmGetAttribute call with the attribute MML_GDC_SURF_ATTR_SURF_FORMAT will return the related color format if the same attributes match the MML_GDC_SURF_FORMAT definition.*

MML_GDC_SURF_ATTR_USERDEFINED

User defined (initial: 0).

11.4.4.2 enum MML_GDC_SURF_CLF

Color format of color lookup table.

Enumerator

MML_GDC_SURF_CLF_R8G8B8
R8G8B8

MML_GDC_SURF_CLF_B8G8R8
B8G8R8

MML_GDC_SURF_CLF_R5G5B5
R5G5B5

MML_GDC_SURF_CLF_A1R5G5B5
A1R5G5B5

MML_GDC_SURF_CLF_A4R4G4B4
A4R4G4B4

11.4.4.3 enum MML_GDC_SURF_CLM

Mode definition for color lookup table.

Enumerator

MML_GDC_SURF_CLM_NEUTRAL
Module in neutral mode, input data is bypassed to the output.

MML_GDC_SURF_CLM_INDEX_RGB
Module in color index table mode (LUT holds a R, G, B color value, indexed with the red input color).

MML_GDC_SURF_CLM_INDEX_RGBA
Module in color index table mode (LUT holds a R, G, B, A color value, indexed with the red input color).

11.4.4.4 enum MML_GDC_SURF_COMP

Compression format.

Enumerator

MML_GDC_SURF_COMP_NON
The buffer is not compressed.

MML_GDC_SURF_COMP_RLC
Run-Length Encoded (allowed for read buffers only).

MML_GDC_SURF_COMP_RLA
Run-Length Adaptive (lossless compression, allowed for read buffers only).

MML_GDC_SURF_COMP_RLAD
Run-Length Adaptive Dithering (lossy compression).

11.4.4.5 enum MML_GDC_SURF_FORMAT

Color format of surface buffer. The syntax for RGBA buffers is the following: R, G, B, A and X stands for red, green, blue, alpha and unused. The field description(s) is followed by the bit width. For instance R5G6B5 used 5 red, 6 green and 5 blue bits but no alpha.

Note:

- *Additional formats are supported by the PixEng hardware. They can be defined with the attribute function.*

The following examples show the related memory organization:

R8G8B8A8:

Byte	0	1	2	3	4	5	6	7
Color	A0	B0	G0	R0	A1	B1	G1	R1

RGB8A8:

Byte	0	1	2	3
Color	A0	RGB0	A1	RGB1

The memory organisation is described with below.

Enumerator

MML_GDC_SURF_FORMAT_R8G8B8A8
32 bpp RGBA format.

MML_GDC_SURF_FORMAT_A8B8G8R8
32 bpp ABGR format.

MML_GDC_SURF_FORMAT_A8R8G8B8
32 bpp ARGB format.

MML_GDC_SURF_FORMAT_B8G8R8A8
32 bpp BGRA format.

MML_GDC_SURF_FORMAT_R8G8B8X8
32 bpp RGB format.

MML_GDC_SURF_FORMAT_X8B8G8R8
32 bpp BGR format.

MML_GDC_SURF_FORMAT_X8R8G8B8
32 bpp RGB format.

MML_GDC_SURF_FORMAT_R8G8B8
24 bpp RGB format.

MML_GDC_SURF_FORMAT_B8G8R8
24 bpp BGR format.

MML_GDC_SURF_FORMAT_R6G6B6
18 bpp BGR format.

MML_GDC_SURF_FORMAT_R4G4B4A4
16 bpp RGBA format.

MML_GDC_SURF_FORMAT_A4R4G4B4
16 bpp ARGB format.

MML_GDC_SURF_FORMAT_R5G5B5A1
16 bpp RGBA format (5 bit for RGB, 1 bit alpha).

MML_GDC_SURF_FORMAT_A1R5G5B5
16 bpp ARGB format (5 bit for RGB, 1 bit alpha).

MML_GDC_SURF_FORMAT_A1B5G5R5
16 bpp ABGR format (5 bit for RGB, 1 bit alpha).

MML_GDC_SURF_FORMAT_B5G5R5A1
16 bpp BGRA format (5 bit for RGB, 1 bit alpha).

MML_GDC_SURF_FORMAT_R5G6B5
16 bpp BGR format (5 bit for RB, 6 bit for G).

MML_GDC_SURF_FORMAT_A8RGB8
16 bpp, A8RGB8 can be used for gray or indexed image buffers with additional alpha value. For the second use case an indexed color lookup table must be defined in the surface.

MML_GDC_SURF_FORMAT_RGB8
8 bpp, RGB8 can be used for gray or indexed image buffers. For the second use case an indexed color lookup table must be defined in the surface.

MML_GDC_SURF_FORMAT_A8
8 bpp alpha format, can be used (e.g., as text buffer or external alpha mask buffer).

MML_GDC_SURF_FORMAT_A4RGB4
8 bpp, A4RGB4 can be used for gray or indexed image buffers with additional alpha value. For the second use case an indexed color lookup table must be defined in the surface.

MML_GDC_SURF_FORMAT_A4
4 bpp alpha format, can be used (e.g., as text buffer or external alpha mask buffer).

MML_GDC_SURF_FORMAT_A2
2 bpp alpha format, can be used (e.g., as text buffer or external alpha mask buffer).

MML_GDC_SURF_FORMAT_A1
1 bpp alpha format, can be used (e.g., as text buffer or external alpha mask buffer).

MML_GDC_SURF_FORMAT_RGB1
1 bpp back/white buffer (no alpha).

11.4.5 Function Documentation

11.4.5.1 MM_ERROR mmlGdcSmAssignBuffer(MML_GDC_SURFACE surf, MM_U32 uWidth, MM_U32 uHeight, MML_GDC_SURF_FORMAT eFormat, void * pBufferAddress, MM_U32 uRleWords)

Assign a memory address, width, height and color format representing an image to a surface object. The buffer is owned by the calling function. It just describes how the image buffer must be used by a function. The application must ensure that the memory is available as long as the surface is being used.

Note:

- *The mmlGdcSmAssignBuffer call is a fast way to assign an image to a surface object. Alternatively it is also possible to assign the same properties with several calls of mmlGdcSmSetAttribute.*
- *The eFormat value can be used to define the most useful color formats. Please note that not all hardware units support all color formats. The MML_GDC_SURF_FORMAT description includes hints which format can be used with which unit.*
- *The PixEng HW can operate with many more color formats. Use mmlGdcSmSetAttribute in this case to assign the correct format to the surface.*

- *In some cases pBufferAddress can be zero. Such surfaces can be used as source surfaces in PixEng operations. In this case the hardware will not access surface memory but the driver uses the geometry settings of the surface.*
- *Surfaces with run length encoded buffers (uRleWords != 0) are only supported for source surfaces in PixEng operations. If uRleWords is different from zero MML_GDC_SURF_COMP_RLC will be set to MML_GDC_SURF_ATTR_COMPRESSION_FORMAT, otherwise MML_GDC_SURF_COMP_NON.*

Parameters

in	surf	The surface object.
in	uWidth	The width in pixels of the image.
in	uHeight	The height in pixels of the image.
in	eFormat	The format of the image. The format defines the fields BitPerPixel, ColorBits, ColorShift, Color format.
in	pBufferAddress	The memory address of the image. The buffer starts with the upper left pixel.
in	uRleWords	Number of 32-bit words that are required to decode the run length encoded source buffer. Zero indicates an uncompressed buffer.

Return values

MML_OK	On success.
MML_ERR_GDC_SURF_INVALID_SURFACE	If NULL pointer is given for surf.
MML_ERR_GDC_SURF_INVALID_FORMAT	If illegal value is given for eFormat.
MML_ERR_GDC_SURF_INVALID_PARAMETER	If surface size is out of range, see MML_GDC_SURFACE_MAX_WIDTH and MML_GDC_SURFACE_MAX_HEIGHT.

11.4.5.2 MM_ERROR mmlGdcSmAssignClut(MML_GDC_SURFACE surf, MML_GDC_SURF_CLM eMode, MM_U32 uCount, MML_GDC_SURF_CLF eFormat, void * pBufferAddress)

Assign a color lookup table to a surface. It points to a VRAM memory address owned by the application. The application must ensure that the memory is available as long as the surface is still in use and that the memory block is large enough with respect to the width, height and format parameters of the surface.

Note:

- *uCount = 0 or pBufferAddress = 0 set the CLUT in neutral mode, input data is bypassed to the output.*

Parameters

in	surf	The surface object getting this new property.
in	eMode	Defines the operation mode for the CLUT.
in	uCount	Number of table entries to be written.
in	eFormat	Format of the table entries.
in	pBufferAddress	The address of the color index buffer.

Return values

MML_OK	On success.
MML_ERR_GDC_SURF_INVALID_SURFACE	If NULL pointer is given for surf.
MML_ERR_GDC_SURF_INVALID_PARAMETER	If illegal value is given for eMode.
MML_ERR_GDC_SURF_INVALID_FORMAT	If illegal value is given for eFormat.

11.4.5.3 MM_ERROR mmlGdcSmGetAttribute(const MML_GDC_SURFACE surf, MML_GDC_SURF_ATTR eName, MM_U32 * puValue)

Get surface attributes.

Parameters

in	surf	The surface.
in	eName	Name of the attribute. See MML_GDC_SURF_ATTR.
out	puValue	Pointer to a variable to receive the parameter value.

Return values

MML_OK	On success.
MML_ERR_GDC_SURF_INVALID_ATTRIBUTE	If illegal value is given for eName.
MML_ERR_GDC_SURF_INVALID_SURFACE	If NULL pointer is given for surf.

11.4.5.4 MM_ERROR mmlGdcSmResetSurfaceObject(MML_GDC_SURFACE surf)

Reset a surface object with default values.

Parameters

in,out	surf	The surface to reset.

Return values

MML_OK	On success, otherwise the related error code.

11.4.5.5 MM_ERROR mmlGdcSmSetAttribute(const MML_GDC_SURFACE surf, MML_GDC_SURF_ATTR eName, MM_U32 uValue)

Set surface attributes. The application must ensure that the parameters like stride, height, size and format of the surface are always consistent and match the size of the memory block allocated for the surface.

Parameters

in	surf	The surface object.
in	eName	Name of the attribute. See MML_GDC_SURF_ATTR.
in	uValue	The new value.

Return values

MML_OK	On success.
MML_ERR_GDC_SURF_INVALID_ATTRIBUTE	If illegal value is given for eName.
MML_ERR_GDC_SURF_INVALID_SURFACE	If NULL pointer is given for surf.

11.5 Display API

The Display API exposes all the hardware features of the display unit. See also Display Overview.

Data Structures

- struct MML_GDC_DISP_MODE_LINE
- struct MML_GDC_DISP_TCON_PROPERTIES
- struct MML_GDC_DISP_PROPERTIES
- struct MML_GDC_DISP_WINDOW_PROPERTIES

Typedefs

- typedef struct MML_GDC_DISPLAY * MML_GDC_DISPLAY
- typedef struct MML_GDC_DISP_WINDOW * MML_GDC_DISP_WINDOW

Enumerations

- enum MML_GDC_DISP_CONTROLLER {
  MML_GDC_DISP_CONTROLLER_0 = 0
  }
- enum MML_GDC_DISP_MODE {
  MML_GDC_DISP_SINGLE_SCREEN = 0,
  MML_GDC_DISP_DUAL_SCREEN,
  MML_GDC_DISP_DUAL_VIEW
  }
- enum MML_GDC_DISP_OUTPUT_SCREEN {
  MML_GDC_DISP_OUTPUT_SCREEN_PRIMARY = 0,
  MML_GDC_DISP_OUTPUT_SCREEN_SECONDARY,
  MML_GDC_DISP_OUTPUT_SCREEN_BOTH
  }
- enum MML_GDC_DISP_FILTER {
  MML_GDC_DISP_FILTER_NEAREST = 0,
  MML_GDC_DISP_FILTER_BILINEAR
  }
- enum MML_GDC_DISP_TILE_MODE {
  MML_GDC_DISP_TILE_MODE_ZERO = 0,
  MML_GDC_DISP_TILE_MODE_CONST = 1,
  MML_GDC_DISP_TILE_MODE_PAD = 2,
  MML_GDC_DISP_TILE_MODE_CLIP = 3
  }
- enum MML_GDC_DISP_LAYER {
  MML_GDC_DISP_LAYER_0 = 0,
  MML_GDC_DISP_LAYER_1
  }
- enum MML_GDC_DISP_SUB_LAYER {
  MML_GDC_DISP_SUB_LAYER_DEFAULT = 0,
  MML_GDC_DISP_SUB_LAYER_1,
  MML_GDC_DISP_SUB_LAYER_2,
  MML_GDC_DISP_SUB_LAYER_3,
  MML_GDC_DISP_SUB_LAYER_4,
  MML_GDC_DISP_SUB_LAYER_5,
  MML_GDC_DISP_SUB_LAYER_6,
  MML_GDC_DISP_SUB_LAYER_7,

```
        MML_GDC_DISP_SUB_LAYER_8
    }
- enum MML_GDC_DISP_DCK_DELAY_ENABLE {
        MML_GDC_DISP_DCK_DELAY_OFF = 0,
        MML_GDC_DISP_DCK_DELAY_ON
    }
- enum MML_GDC_DISP_DCK_INVERT_ENABLE {
        MML_GDC_DISP_DCK_INVERT_OFF = 0,
        MML_GDC_DISP_DCK_INVERT_ON
    }
- enum MML_GDC_DISP_DITHER_ENABLE {
        MML_GDC_DISP_DITHOFF = 0,
        MML_GDC_DISP_DITHON
    }
- enum MML_GDC_DISP_DITHER_MODE {
        MML_GDC_DISP_TEMPDITH = 0,
        MML_GDC_DISP_SPATDITH = (1 << 4)
    }
- enum MML_GDC_DISP_DITHER_RANGE {
        MML_GDC_DISP_DITHRS11LOW = 0
    }
- enum MML_GDC_DISP_DITHER_FORMAT {
        MML_GDC_DISP_DITHER108 = 0x08080800,
        MML_GDC_DISP_DITHER107 = 0x07070700,
        MML_GDC_DISP_DITHER106 = 0x06060600,
        MML_GDC_DISP_DITHER105 = 0x05060500
    }
- enum MML_GDC_DISP_CLUT_FORMAT {
        MML_GDC_DISP_CLUT_FORMAT_33 = 33
    }
- enum MML_GDC_DISP_CMATRIX_FORMAT {
        MML_GDC_DISP_CMATRIX_FORMAT_4X3 = 0,
        MML_GDC_DISP_CMATRIX_FORMAT_5X4
    }
- enum MML_GDC_DISP_ATTR {
        MML_GDC_DISP_ATTR_OUTPUT_CONTROLLER = 0,
        MML_GDC_DISP_ATTR_X_RESOLUTION,
        MML_GDC_DISP_ATTR_Y_RESOLUTION,
        MML_GDC_DISP_ATTR_BUFF_ERR,
        MML_GDC_DISP_ATTR_BACKGROUND_COLOR
    }
- enum MML_GDC_DISP_WIN_ATTR {
        MML_GDC_DISP_WIN_ATTR_LAYER_ID = 0,
        MML_GDC_DISP_WIN_ATTR_SUB_LAYER_ID,
        MML_GDC_DISP_WIN_ATTR_TOPLEFT_X,
        MML_GDC_DISP_WIN_ATTR_TOPLEFT_Y,
        MML_GDC_DISP_WIN_ATTR_WIDTH,
        MML_GDC_DISP_WIN_ATTR_HEIGHT,
        MML_GDC_DISP_WIN_ATTR_SCREEN,
        MML_GDC_DISP_WIN_ATTR_COLOR,
        MML_GDC_DISP_WIN_ATTR_DISABLE,
        MML_GDC_DISP_WIN_ATTR_SWAP_INTERVAL,
        MML_GDC_DISP_WIN_ATTR_MAX_BUFFER,
```

```
        MML_GDC_DISP_WIN_ATTR_TILE_MODE,
        MML_GDC_DISP_WIN_ATTR_FEATURE
    }
```

Layer feature request

- #define MML_GDC_DISP_FEATURE_INDEX_COLOR (1 << 0)
- #define MML_GDC_DISP_FEATURE_DECODE (1 << 1)
- #define MML_GDC_DISP_FEATURE_MULTI_LAYER (1 << 7)

Buffer target

- #define MML_GDC_DISP_BUFF_TARGET_COLOR_BUFF (1 << 1)

Blend modes

- #define MML_GDC_DISP_BLEND_NONE (0)
- #define MML_GDC_DISP_BLEND_TRANSPARENCY (1U << 0)
- #define MML_GDC_DISP_BLEND_GLOBAL_ALPHA (1U << 1)
- #define MML_GDC_DISP_BLEND_SOURCE_ALPHA (1U << 2)
- #define MML_GDC_DISP_BLEND_SOURCE_MULTIPLY_ALPHA (1U << 4)

Polarity control.

- #define MML_GDC_DISP_HSYNC_LOW (0)
- #define MML_GDC_DISP_HSYNC_HIGH (1U << 0)
- #define MML_GDC_DISP_VSYNC_LOW (0)
- #define MML_GDC_DISP_VSYNC_HIGH (1U << 1)
- #define MML_GDC_DISP_DE_LOW (0)
- #define MML_GDC_DISP_DE_HIGH (1U << 2)
- #define MML_GDC_DISP_RGB_LOW (0)
- #define MML_GDC_DISP_RGB_HIGH (1U << 3)

Default initializer

- #define MML_GDC_DISP_PROPERTIES_INITIALIZER
- #define MML_GDC_DISP_WINDOW_PROPERTIES_INITIALIZER

Display Functions

- MM_ERROR mmlGdcDispOpenDisplay (MML_GDC_DISP_PROPERTIES ∗mode, MML_GDC_DISPLAY ∗display)
- MM_ERROR mmlGdcDispCloseDisplay (MML_GDC_DISPLAY display)
- MM_ERROR mmlGdcDispDitherCtrl (MML_GDC_DISPLAY display, MML_GDC_DISP_DITHER_ENABLE enable, MML_GDC_DISP_DITHER_MODE mode, MML_GDC_DISP_DITHER_RANGE range, MML_GDC_DISP_DITHER_FORMAT format)
- MM_ERROR mmlGdcDispCLUTData (MML_GDC_DISPLAY display, MML_GDC_DISP_CLUT_FORMAT format, const MM_S16 ∗pRed, const MM_S16 ∗pGreen, const MM_S16 ∗pBlue)
- MM_ERROR mmlGdcDispSyncVSync (MML_GDC_DISPLAY display, MML_GDC_SYNC sync, MM_S32 vsyncCnt)
- MM_ERROR mmlGdcDispSetAttribute (MML_GDC_DISPLAY display, MML_GDC_DISP_ATTR pname, MM_U32 param)
- MM_ERROR mmlGdcDispGetAttribute (MML_GDC_DISPLAY display, MML_GDC_DISP_ATTR pname, MM_U32 ∗pParam)

- MM_ERROR mmlGdcDispCommit (MML_GDC_DISPLAY display)

Window Functions

- MM_ERROR mmlGdcDispWinCreate (MML_GDC_DISPLAY display, MML_GDC_DISP_WINDOW_PROPERTIES ∗properties, MML_GDC_DISP_WINDOW ∗pWin)
- MM_ERROR mmlGdcDispWinDestroy (MML_GDC_DISP_WINDOW win)
- MM_ERROR mmlGdcDispWinSetSurface (MML_GDC_DISP_WINDOW win, MM_U32 target, MML_GDC_SURFACE surf)
- MM_ERROR mmlGdcDispWinSetBlendMode (MML_GDC_DISP_WINDOW win, MM_U32 blend_mode)
- MM_ERROR mmlGdcDispWinSetMatrix (MML_GDC_DISP_WINDOW win, MM_U32 target, const MM_FLOAT ∗matrix)
- MM_ERROR mmlGdcDispWinSync (MML_GDC_DISP_WINDOW win, MML_GDC_SYNC sync)
- MM_ERROR mmlGdcDispWinWaitSync (MML_GDC_DISP_WINDOW win, MML_GDC_SYNC sync)
- MM_ERROR mmlGdcDispWinSetAttribute (MML_GDC_DISP_WINDOW win, MML_GDC_DISP_WIN_ATTR pname, MM_U32 param)
- MM_ERROR mmlGdcDispWinGetAttribute (MML_GDC_DISP_WINDOW win, MML_GDC_DISP_WIN_ATTR pname, MM_U32 ∗pParam)
- MM_ERROR mmlGdcDispWinCommit (MML_GDC_DISP_WINDOW win)

11.5.1 Detailed Description

The Display API exposes all the hardware features of the display unit. See also Display Overview.

#include "mml_gdc_display.h"

The software interface provides 2 objects required to use and control the display unit:

The MML_GDC_DISPLAY is a software handle for a hardware display controller and is required to

- Set up video modes.
- Configure dithering or gamma correction.

The MML_GDC_DISP_WINDOW is the software handle for hardware layers and is required

- To use hardware layers to show rendered content.
- To configure blending and transparency of layers.
- For synchronization between layers and other hardware components.

The following example demonstrates the steps to show an image on Display 0:

```
// Use default initializer for the properties and change later the important fields.
MML_GDC_DISP_PROPERTIES          dispProp     = MML_GDC_DISP_PROPERTIES_INITIALIZER;
MML_GDC_DISP_WINDOW_PROPERTIES   windowProp =
MML_GDC_DISP_WINDOW_PROPERTIES_INITIALIZER;
MML_GDC_DISPLAY                  display;
MML_GDC_DISP_WINDOW              win;
MML_GDC_SURFACE                  target;

// Set  your  requested  display  properties.
```

```
dispProp.outputController = MML_GDC_DISP_CONTROLLER_0;
dispProp.xResolution   = 640;
dispProp.yResolution = 480;
// Open the display
mmlGdcDispOpenDisplay( &dispProp, &display);
// Set Window properties.
windowProp.topLeftX = 0;
windowProp.topLeftY = 0;
windowProp.width       = 640;
windowProp.height     = 480;
//Create the window.
mmlGdcDispWinCreate(display, &windowProp, &win);
// Draw something in a surface.
MyDrawFunction(target);
// Push the surface to the surface to the window.
mmlGdcDispWinSetSurface(win, target);
mmlGdcDispWinCommit(win);

//Close Window and Display.
mmlGdcDispWinDestroy(win);
mmlGdcDispCloseDisplay(display);
```

Like mentioned in the Display Overview enhanced features (MML_GDC_DISP_WINDOW_PROPERTIES) can be requested while opening (mmlGdcDispWinCreate) a window. The table below lists the available feature types and there restrictions:

Window Feature	Comment	Restrictions
MML_GDC_DISP_FEATURE_INDEX_COLOR	The window can show an indexed image.	
MML_GDC_DISP_FEATURE_DECODE	The window can display a RLE or RLAD buffer.	– The surface must not be mirrored, rotated if compression is used. – Cannot be combined with MML_GDC_DISP_FEATURE_MULTI_LAYER. – Only 2 windows with this feature are available.

Window Feature	Comment	Restrictions
MML_GDC_DISP_FEATURE_MULTI_LAYER	Up to 8 windows with different size, color format and buffer address but the same layerId can be opened and used simultaneously for one display. The combined windows represent a common layer that can be blended to the lower level windows. For overlapping windows ID the resulting pixel is defined by the latest opened window.	– Overlapping windows of this layer cannot be blended one on top of the other. Only the top most window will be blended against the background. – Cannot be combined with MML_GDC_DISP_FEATURE_DECODE.

11.5.2 Macro Definition Documentation

11.5.2.1 #define MML_GDC_DISP_BLEND_GLOBAL_ALPHA (1U << 1)

Enable global alpha blending.

11.5.2.2 #define MML_GDC_DISP_BLEND_NONE (0)

Disable blending.

11.5.2.3 #define MML_GDC_DISP_BLEND_SOURCE_ALPHA (1U << 2)

Enable per pixel source alpha blending.

11.5.2.4 #define MML_GDC_DISP_BLEND_SOURCE_MULTIPLY_ALPHA (1U << 4)

Enable source alpha multiplication.

11.5.2.5 #define MML_GDC_DISP_BLEND_TRANSPARENCY (1U << 0)

Enable transparency.

11.5.2.6 #define MML_GDC_DISP_BUFF_TARGET_COLOR_BUFF (1 << 1)

Color buffer as target buffer.

11.5.2.7 #define MML_GDC_DISP_DE_HIGH (1U << 2)

Data enable signal high active.

11.5.2.8 #define MML_GDC_DISP_DE_LOW (0)

Data enable signal low active.

11.5.2.9 #define MML_GDC_DISP_FEATURE_DECODE (1 << 1)

Show encoded images.

11.5.2.10 #define MML_GDC_DISP_FEATURE_INDEX_COLOR (1 << 0)

Indexed color support .

11.5.2.11 #define MML_GDC_DISP_FEATURE_MULTI_LAYER (1 << 7)

The window is a "Multi-Window", (i.e., it shares the same layer with all other "Multi-Windows").

11.5.2.12 #define MML_GDC_DISP_HSYNC_HIGH (1U << 0)

Hsync signal high active.

11.5.2.13 #define MML_GDC_DISP_HSYNC_LOW (0)

Hsync signal low active.

11.5.2.14 #define MML_GDC_DISP_PROPERTIES_INITIALIZER

Value:

{ MML_GDC_DISP_CONTROLLER_0, MML_GDC_DISP_SINGLE_SCREEN, 0, 0, 60, 0, 0, 0, 0 }

11.5.2.15 #define MML_GDC_DISP_RGB_HIGH (1U << 3)

Pixel data inverted.

11.5.2.16 #define MML_GDC_DISP_RGB_LOW (0)

No inversion of pixel data.

11.5.2.17 #define MML_GDC_DISP_VSYNC_HIGH (1U << 1)

Vsync signal high active.

11.5.2.18 #define MML_GDC_DISP_VSYNC_LOW (0)

Vsync signal low active.

11.5.2.19 #define MML_GDC_DISP_WINDOW_PROPERTIES_INITIALIZER

Value:

{ MML_GDC_DISP_OUTPUT_SCREEN_PRIMARY, 0, 0, 0, 0, MML_GDC_DISP_LAYER_0, 0, MML_GDC_DISP_SUB_LAYER_DEFAULT}

11.5.3 Typedef Documentation

11.5.3.1 typedef struct MML_GDC_DISP_WINDOW *MML_GDC_DISP_WINDOW

Window object.

11.5.3.2 typedef struct MML_GDC_DISPLAY * MML_GDC_DISPLAY

Display object.

11.5.4 Enumeration Type Documentation

11.5.4.1 enum MML_GDC_DISP_ATTR

Enumeration of the different configuration attributes for display controllers.

Enumerator

MML_GDC_DISP_ATTR_OUTPUT_CONTROLLER
Display controller used for the display (see MML_GDC_DISP_CONTROLLER). This attribute can only be read.

MML_GDC_DISP_ATTR_X_RESOLUTION
Horizontal resolution. This attribute can only be read.

MML_GDC_DISP_ATTR_Y_RESOLUTION
Vertical resolution. This attribute can only be read.

MML_GDC_DISP_ATTR_BUFF_ERR
If internal response time to read SDRAM is too long, internal FIFO buffer fails to supply display data. This attribute shows error status of the FIFO for this display controller. This attribute can only be read. The hardware status is cleared after read operation.

- Get value = 0, no buffer error.
- Get value != 0, buffer error occurred.

MML_GDC_DISP_ATTR_BACKGROUND_COLOR
Sets background color 0xRRGGBBAA for the screen area not included in any display window. Background color is default blended with display layers. The default value is 0 (black).

11.5.4.2 enum MML_GDC_DISP_CLUT_FORMAT

For size of entries for the CLUT.

Enumerator

MML_GDC_DISP_CLUT_FORMAT_33
Each array for RGB contains 33 10-bit values to describe the 0-255 index range. The missing values are interpolated (see mmlGdcDispCLUTData for details).

11.5.4.3 enum MML_GDC_DISP_CMATRIX_FORMAT

Color matrix format.

Enumerator

MML_GDC_DISP_CMATRIX_FORMAT_4X3
float[12] array with 4 column and 3 lines.

MML_GDC_DISP_CMATRIX_FORMAT_5X4
float[20] array with 5 column and 4 lines.

11.5.4.4 enum MML_GDC_DISP_CONTROLLER

Enumeration of display controllers.

Enumerator

MML_GDC_DISP_CONTROLLER_0
Display controller 0.

11.5.4.5 enum MML_GDC_DISP_DCK_DELAY_ENABLE

Display clock delay disable/enable flags.

Enumerator

MML_GDC_DISP_DCK_DELAY_OFF
Disable the display clock delay.

MML_GDC_DISP_DCK_DELAY_ON
Enable the display clock delay.

11.5.4.6 enum MML_GDC_DISP_DCK_INVERT_ENABLE

Inversion of display clock disable/enable flags.

Enumerator

MML_GDC_DISP_DCK_INVERT_OFF
Display clock output signal is not inverted.

MML_GDC_DISP_DCK_INVERT_ON
Display clock output signal is inverted.

11.5.4.7 enum MML_GDC_DISP_DITHER_ENABLE

Dither enable.

Enumerator

MML_GDC_DISP_DITHOFF
Flag to disable dithering.

MML_GDC_DISP_DITHON
Flag to enable dithering.

11.5.4.8 enum MML_GDC_DISP_DITHER_FORMAT

Dither format 0x0R0G0B00.

Enumerator

MML_GDC_DISP_DITHER108
Flag to specify dithering output format of RGB 10x10x10 -> 8x8x8.

MML_GDC_DISP_DITHER107
Flag to specify dithering output format of RGB 10x10x10 -> 7x7x7.

MML_GDC_DISP_DITHER106
Flag to specify dithering output format of RGB 10x10x10 -> 6x6x6.

MML_GDC_DISP_DITHER105
Flag to specify dithering output format of RGB 10x10x10 -> 5x6x5.

11.5.4.9 enum MML_GDC_DISP_DITHER_MODE

Dither mode.

Enumerator

MML_GDC_DISP_TEMPDITH
Flag to specify temporal dithering.

MML_GDC_DISP_SPATDITH
Flag to specify spatial dithering.

11.5.4.10 enum MML_GDC_DISP_DITHER_RANGE

Dither range.

Enumerator

MML_GDC_DISP_DITHRS11LOW
Flag to specify dither range: add 0s to lower bits.

11.5.4.11 enum MML_GDC_DISP_FILTER

Enumeration of possible filter settings for a window.

Enumerator

MML_GDC_DISP_FILTER_NEAREST
Nearest filter enable.

MML_GDC_DISP_FILTER_BILINEAR
Bilinear filter enable.

11.5.4.12 enum MML_GDC_DISP_LAYER

Enumeration of layers.

Enumerator

MML_GDC_DISP_LAYER_0
Layer 0

MML_GDC_DISP_LAYER_1
Layer 1

11.5.4.13 enum MML_GDC_DISP_MODE

Enumeration of display modes.

Enumerator

MML_GDC_DISP_SINGLE_SCREEN
Single screen mode.

MML_GDC_DISP_DUAL_SCREEN
Reserved for future use.

MML_GDC_DISP_DUAL_VIEW
Reserved for future use.

11.5.4.14 enum MML_GDC_DISP_OUTPUT_SCREEN

Enumeration of possible locations to show a layer on a display.

Enumerator

MML_GDC_DISP_OUTPUT_SCREEN_PRIMARY
Show layer on primary screen.

MML_GDC_DISP_OUTPUT_SCREEN_SECONDARY
Show layer on secondary screen (implies using dual screen mode see MML_GDC_DISP_PROPERTIES).

MML_GDC_DISP_OUTPUT_SCREEN_BOTH
Show layer on both screens (implies using dual screen mode see MML_GDC_DISP_PROPERTIES).

11.5.4.15 enum MML_GDC_DISP_SUB_LAYER

Enumeration of sub-layers for windows with feature MML_GDC_DISP_FEATURE_MULTI_LAYER.

Enumerator

MML_GDC_DISP_SUB_LAYER_DEFAULT
Window is not a Multi Window or the next free sub-layer is used.

MML_GDC_DISP_SUB_LAYER_1
sub layer 1

MML_GDC_DISP_SUB_LAYER_2
sub layer 2

MML_GDC_DISP_SUB_LAYER_3
sub layer 3

MML_GDC_DISP_SUB_LAYER_4
sub layer 4

MML_GDC_DISP_SUB_LAYER_5
sub layer 5

MML_GDC_DISP_SUB_LAYER_6
sub layer 6

MML_GDC_DISP_SUB_LAYER_7
sub layer 7

MML_GDC_DISP_SUB_LAYER_8
sub layer 8

11.5.4.16 enum MML_GDC_DISP_TILE_MODE

Enumeration of possible tile modes for a window.

Enumerator

MML_GDC_DISP_TILE_MODE_ZERO
Pixel outside the surface are 0.

MML_GDC_DISP_TILE_MODE_CONST
Pixel outside the surface use the const color of the window.

MML_GDC_DISP_TILE_MODE_PAD
Pixel outside the surface use the closest pixel from source buffer, this must not be set for RLD operations

MML_GDC_DISP_TILE_MODE_CLIP
The window position and size will be clipped to the overlapped area of the given window and the surface.

11.5.4.17 enum MML_GDC_DISP_WIN_ATTR

Enumeration of the different configuration attributes for windows.

Enumerator

MML_GDC_DISP_WIN_ATTR_LAYER_ID

Layer used for the window (see MML_GDC_DISP_LAYER). This attribute can only be read.

MML_GDC_DISP_WIN_ATTR_SUB_LAYER_ID

Sub layer used for the window (MML_GDC_DISP_SUB_LAYER_1 .. MML_GDC_DISP_SUB_LAYER_8) or MML_GDC_DISP_SUB_LAYER_DEFAULT if feature MML_GDC_DISP_FEATURE_MULTI_LAYER is not used for the window. This attribute can only be read.

MML_GDC_DISP_WIN_ATTR_TOPLEFT_X

Top left X coordinate of the window on the display.

Note:

- *To set a negative value for X use the following formula:*
  value = 0xffffffff - ((MM_U32)(-X) - 1);
  If the value is returned by mmlGdcDispWinGetAttribute, the most significant bit must be used to check for negative values:
  X = ((value & 0x80000000) == 0) ? (int)value : -(int)((0xffffffff - value) + 1);

MML_GDC_DISP_WIN_ATTR_TOPLEFT_Y

Top left Y coordinate of the window on the display.

Note:

- *The Y value can be negative. Description see MML_GDC_DISP_WIN_ATTR_TOPLEFT_X.*

MML_GDC_DISP_WIN_ATTR_WIDTH

Width of window on the display.

Note:

- *The area beyond the range of the underlying framebuffer or surface will be filled as black.*

MML_GDC_DISP_WIN_ATTR_HEIGHT

Height of window on the display.

Note:

- *The area beyond the range of the underlying framebuffer or surface will be filled as black.*

MML_GDC_DISP_WIN_ATTR_SCREEN

Select the screen(s), where the layer is displayed. Alpha layers do not have this attribute. See MML_GDC_DISP_OUTPUT_SCREEN.

- MML_GDC_DISP_OUTPUT_SCREEN_PRIMARY = Show layer on screen 0.
- MML_GDC_DISP_OUTPUT_SCREEN_SECONDARY = Show layer on screen 1.
- MML_GDC_DISP_OUTPUT_SCREEN_BOTH = Show layer on both screens.

MML_GDC_DISP_WIN_ATTR_COLOR

Set window color. The format of the color value is 0xRRGGBBAA. Three use cases are possible for the window color.

- If the color surface set to the window has no RGB color, the surface fetches the RGB color from the window color. The window color will be ignored if the surface brings the RGB color by itself.
- If blend mode is MML_GDC_DISP_BLEND_TRANSPARENCY, the transparency color is defined by the RGB part of the window color.
- If blend mode is MML_GDC_DISP_BLEND_GLOBAL_ALPHA, the global alpha value is defined by the alpha part of the window color.

MML_GDC_DISP_WIN_ATTR_DISABLE

Switch the window off.

- Default value = 0, window is enabled.
- Set value = 1, window is switched off and invisible on screen.

MML_GDC_DISP_WIN_ATTR_SWAP_INTERVAL

Selects the swap interval to be used for displaying sur- faces. This will be used if different surfaces get shown after each other using mmlGdcDispWinSetSurface and mmlGdcDispWinCommit. The default value = 1.

The minimum allowed value = 1 (0 is possible but results in display flicker). The maximum allowed value = (2∧31)-1.

MML_GDC_DISP_WIN_ATTR_MAX_BUFFER

Defines the maximum number of framebuffers that can be queued by the driver for a window. If an application submits surfaces using mmlGdcDispWinSetSurface and mmlGdcDispWinCommit faster than they can be displayed, the driver queues them up to the maximum specified by MML_GDC_DISP_WIN_ATTR_MAX_BUFFER.

This attribute can only be read.

MML_GDC_DISP_WIN_ATTR_TILE_MODE

This attribute can be used to define the tiling mode for windows.

The tile mode defines the color of pixels outside the surface but inside the window. This is relevant if the assigned surface is smaller than the window or the geometry matrix for the window if moves the surface out of the window. The tile mode must be a value of MML_GDC_DISP_TILE_MODE. The default setting is MML_GDC_DISP_TILE_MODE_CLIP.

Note:

- *If MML_GDC_DISP_TILE_MODE_CONST is set for a window without an attached surface than the const color fills the window area.*

Tip:

- *The mode MML_GDC_DISP_TILE_MODE_PAD can be used to generate a gradient background with a single line surface.*

MML_GDC_DISP_WIN_ATTR_FEATURE

This attribute can be used by mmlGdcDispWinGetAttribute only.

It returns the available features for the given windows handle. An application must use the features parameter of the MML_GDC_DISP_WINDOW_PROPERTIES structure to request a window feature when creating the window.

11.5.5 Function Documentation

11.5.5.1 MM_ERROR mmlGdcDispCloseDisplay(MML_GDC_DISPLAY display)

Close a display and all windows opened by this display. By default this function is blocked until previous operations of device display are completely executed. Use mmlGdcConfigSetAttribute(), set MML_GDC_CONFIG_ATTR_DISPLAY_NOBLOCK to 1 to make it non-blocking.

Note:

- *The display closed by the last process switches the display controller off.*

Parameters

in	display	An MML_GDC_DISPLAY returned from a previous call to mmlGdcDispOpenDisplay.

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	If an invalid argument was passed.
MML_ERR_GDC_DISP_DEV_BUSY	If the writing to the device display is denied, because the previous commit, open, create or destroy call is not completely executed (e.g., shadow load request is pending). Call again later!

11.5.5.2 MM_ERROR mmlGdcDispCLUTData(MML_GDC_DISPLAY display, MML_GDC_DISP_CLUT_FORMAT format, const MM_S16 * pRed, const MM_S16 * pGreen, const MM_S16 * pBlue)

Used to configure the color lookup table(CLUT) on the display controller (e.g., for gamma correction).

The format MML_GDC_DISP_CLUT_FORMAT_33 defines 33 sample points representing the resulting color channel intensity. Intermediate values will be interpolated by the HW. The 1st sample point corresponds to input color code 0, 2nd one to 32, ..., last one to 1024 of the 10 bit 2D core internal processing pipeline. Although input 1024 is not possible, the last sample point is needed for interpolation of codes 993 to 1023.

An index entry of 0 stands for the minimum and 1023 for the maximum intensity. Index values outside this range will be clamped.

Note:

- *Example: Let F(in) be the requested gamma formula. Input values of F(in) are in the range [0.0, 1.0]. It is allowed that the output value is smaller than 0.0 or bigger 1.0. The value array (in this example pRed) must be calculated in the following way:*

  MML_GDC_DISP_CLUT_FORMAT_33:

  ```
  for (i = 0; i <= 32; i++)
      pRed[i] = (MM_S16)(0.5f + ( F(i/32.0f * 1024.0f/1023.0f) * 1023));
  ```

  Please note that the given formula calculates the value for F(256/255). If F(x) is only defined for input values 0.0..1.0 then pRed[32] can be calculated as

  ```
  pRed[32] = (MM_S16)(0.5f + (( 32.0f * F(1) - F(31.0f * 32.0f / 1023.0f)) * 1023.0f / 31.0f));
  ```

If one pointer of color components is NULL, then the CLUT is set to bypass.

This setting will not be active immediately. Use mmlGdcDispCommit to submit for processing. The three pointers to array of color component must be valid till the setting is committed.

Parameters

in	display	An MML_GDC_DISPLAY returned from a previous call to mmlGdcDispOpenDisplay.
in	format	Defines the number of entries in the array. Depending on the hardware the CLUT hardware may support not all format types. In this case the driver interpolates the missing or skips the needless values. S6E2D accepts only MML_GDC_DISP_CLUT_FORMAT_33.
in	pRed	Pointer to array of red values. The size of the array depends on format.
in	pGreen	Pointer to array of green values. The size of the array depends on format.
in	pBlue	Pointer to array of blue values. The size of the array depends on format.

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	If an invalid argument was passed.

11.5.5.3 MM_ERROR mmlGdcDispCommit (MML_GDC_DISPLAY display)

The display related setting modification will not be active immediately. The mmlGdcDispCommit submits these settings for processing. By default this function is blocked until previous operations of device display are completely executed. Use mmlGdcConfigSetAttribute(), set MML_GDC_CONFIG_ATTR_DISPLAY_NOBLOCK to 1 to make it non-blocking.

Parameters

in	display	An MML_GDC_DISPLAY returned from a previous call to mmlGdcDispOpenDisplay().

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	If one of the parameters is invalid.
MML_ERR_GDC_DISP_DEV_BUSY	If the writing to the device display is denied, because the previous commit, open, create or destroy call is not completely executed (e.g. shadow load request is pending). Call again later!

11.5.5.4 MM_ERROR mmlGdcDispDitherCtrl (MML_GDC_DISPLAY display, MML_GDC_DISP_DITHER_ENABLE enable, MML_GDC_DISP_DITHER_MODE mode, MML_GDC_DISP_DITHER_RANGE range, MML_GDC_DISP_DITHER_FORMAT format)

Used to configure dithering on the display controller. The dither processing is active if MML_GDC_DISP_DITHON is set. Dithering improves the display images, if the display has less color levels than the original picture. The number of bits per pixel is lowered from the original value e.g. RGB888 to RGB666 with MML_GDC_DISP_DITHER106. The value of lower bits are randomly round up or down based on location of the pixel in the frame (MML_GDC_DISP_SPATDITH). Or, a random vector is generated to address the dither matrix (MML_GDC_DISP_TEMPDITH).

Note:

- *This setting will not be active immediately. Use mmlGdcDispCommit to submit for processing.*

Parameters

in	display	An MML_GDC_DISPLAY returned from a previous call to mmlGdcDispOpenDisplay.
in	enable	Enable Dithering: – MML_GDC_DISP_DITHOFF = Disable dithering. – MML_GDC_DISP_DITHON = Enable dithering.
in	mode	Select mode for dithering: – MML_GDC_DISP_TEMPDITH = Temporal dithering. – MML_GDC_DISP_SPATDITH = Spatial dithering.
in	range	Sets dither range: – MML_GDC_DISP_DITHRS11LOW = adds 0s to lower bits.
in	format	Select output format for dithering: – MML_GDC_DISP_DITHER108 = 10x10x10->8x8x8 – MML_GDC_DISP_DITHER107 = 10x10x10->7x7x7 – MML_GDC_DISP_DITHER106 = 10x10x10->6x6x6 – MML_GDC_DISP_DITHER105 = 10x10x10->5x6x5

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	If an invalid argument was passed.

11.5.5.5 MM_ERROR mmlGdcDispGetAttribute (MML_GDC_DISPLAY display, MML_GDC_DISP_ATTR pname, MM_U32 * pParam)

Gets the value for attribute pname. display specify for which display controller the attribute should be retrieved.

Parameters

in	display	An MML_GDC_DISPLAY returned from a previous call to mmlGdcDispOpenDisplay().
in	pname	Parameter name. See MML_GDC_DISP_ATTR for valid values.
out	pParam	Address where the read value of the attribute is stored.

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	If one of the parameters is invalid.

11.5.5.6 MM_ERROR mmlGdcDispOpenDisplay (MML_GDC_DISP_PROPERTIES * mode, MML_GDC_DISPLAY *display)

Used to open a display. By default this function is blocked until previous operations of device display are completely executed. Use mmlGdcConfigSetAttribute(), set MML_GDC_CONFIG_ATTR_DISPLAY_NOBLOCK to 1 to make it non-blocking.

Note:

- *This function must only be called once for each display output controller.*

Parameters

in	mode	MML_GDC_DISP_PROPERTIES structure describing the desired resolution and display timings.
out	display	On success will contain a valid MML_GDC_DISPLAY.

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	If an invalid argument was passed.
MML_ERR_GDC_DISP_DISPLAY_ALREADY_OPEN	If the display is already opened.
MML_ERR_GDC_DISP_DEV_BUSY	If the writing to the device display is denied, because the previous close call is not completely executed (e.g., shadow load request is pending). Call again later!

11.5.5.7 MM_ERROR mmlGdcDispSetAttribute (MML_GDC_DISPLAY display, MML_GDC_DISP_ATTR pname, MM_U32 param)

Sets the attribute pname to param. display specify for which display controller the attribute should be set.

Note

- *This setting will not be active immediately. Use mmlGdcDispCommit to submit for processing.*

Parameters

in	display	An MML_GDC_DISPLAY returned from a previous call to mmlGdcDispOpenDisplay().
in	pname	Parameter name. See MML_GDC_DISP_ATTR for valid values.
in	param	Value to set for parameter pname.

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	If one of the parameters is invalid.

11.5.5.8 MM_ERROR mmlGdcDispSyncVSync (MML_GDC_DISPLAY display, MML_GDC_SYNC sync, MM_S32 vsyncCnt)

Initializes the sync object sync to get signaled after vsyncCnt VSync's have happened. The VSync is taken from the display controller specified by display.

Parameters

in	display	An MML_GDC_DISPLAY returned from a previous call to mmlGdcDispOpenDisplay().
in	sync	Sync object to initialize with the sync condition.
in	vsyncCnt	Number of VSync's to elapse until the sync object gets signaled. Parameter must be -0x7FFFFFFF < vsyncCnt < 0x7FFFFFFF.

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	If a parameter is invalid.

11.5.5.9 MM_ERROR mmlGdcDispWinCommit (MML_GDC_DISP_WINDOW win)

All window related updates will be written in a work item. mmlGdcDispWinCommit submit the work item of a window for processing. By default this function is blocked until previous operations of device window are completely executed. Use mmlGdcConfigSetAttribute(), set MML_GDC_CONFIG_ATTR_DISPLAY_NOBLOCK to 1 to make it non-blocking.

Parameters

in	win	An MML_GDC_DISP_WINDOW returned from a previous call to mmlGdcDispWinCreate.

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	One of the parameters is invalid.
MML_ERR_GDC_DISP_DEV_BUSY	The writing to the device window is denied, because the previous commit, create or destroy call is not completely executed (e.g., shadow load request is pending). Call again later!

11.5.5.10 MM_ERROR mmlGdcDispWinCreate (MML_GDC_DISPLAY display, MML_GDC_DISP_WINDOW_PROPERTIES * properties, MML_GDC_DISP_WINDOW * pWin)

Used to create a window. By default this function is blocked until previous operations of device display and device window are completely executed. Use mmlGdcConfigSetAttribute(), set MML_GDC_CONFIG_ATTR_DISPLAY_NOBLOCK to 1 to make it non-blocking.

Note:

This function will not initiate any hardware updates. Only the hardware resources are reserved for this window. Updates are applied after the call of mmlGdcDispWinCommit.

It is suggested to create windows in following order:

- *Window with feature MML_GDC_DISP_FEATURE_DECODE.*
- *Window with feature MML_GDC_DISP_FEATURE_MULTI_LAYER.*
- *Window with feature MML_GDC_DISP_FEATURE_INDEX_COLOR or no feature.*

Parameters

in	display	An MML_GDC_DISPLAY returned from a previous call to mmlGdcDispOpenDisplay, identifying the display to create the window on.
in	properties	A pointer to an MML_GDC_DISP_WINDOW_PROPERTIES structure which specifies the properties of the window to create.
out	pWin	On success will contain an MML_GDC_DISP_WINDOW.

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_LAYER_ALREADY_USED	if the specified layer is already in use.
MML_ERR_GDC_DISP_INVALID_ARG	If an invalid argument was passed.
MML_ERR_GDC_DISP_FAILED	If internal error occurred.
MML_ERR_GDC_DISP_DEV_BUSY	If the writing to the device display or device window is denied, because the previous commit, open or destroy call is not completely executed (e.g. shadow load request is pending). Call again later!

11.5.5.11 MM_ERROR mmlGdcDispWinDestroy (MML_GDC_DISP_WINDOW win)

Used to destroy a window. By default this function is blocked until previous operations of device display and device window are completely executed. Use mmlGdcConfigSetAttribute(), set MML_GDC_CONFIG_ATTR_DISPLAY_NOBLOCK to 1 to make it non-blocking.

Parameters

in	win	An MML_GDC_DISP_WINDOW returned from a previous call to mmlGdcDispWinCreate.

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	If an invalid argument was passed.
MML_ERR_GDC_DISP_FAILED	If an unexpected error occurs.
MML_ERR_GDC_DISP_DEV_BUSY	If the writing to the device display or device window is denied, because the previous commit, open or create call is not completely executed (e.g. shadow load request is pending). Call again later!

11.5.5.12 MM_ERROR mmlGdcDispWinGetAttribute (MML_GDC_DISP_WINDOW win, MML_GDC_DISP_WIN_ATTR pname, MM_U32 * pParam)

Gets the value for attribute pname. win specify for which window the attribute should be retrieved.

Parameters

in	win	An MML_GDC_DISP_WINDOW returned from a previous call to mmlGdcDispWinCreate.
in	pname	Parameter name. See MML_GDC_DISP_WIN_ATTR for valid values.
in	pParam	Address where the read value of the attribute is stored.

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	If one of the parameters is invalid.

11.5.5.13 MM_ERROR mmlGdcDispWinSetAttribute(MML_GDC_DISP_WINDOW win, MML_GDC_DISP_WIN_ATTR pname, MM_U32 param)

Sets the attribute pname to param. win specify for which window the attribute should be set.

Note

- *Any attribute settings of the window does not becomes active immediately with the related mmlGdcDispWinSetAttribute call, but will be queued together with other settings of this window. Use mmlGdcDispWinCommit to submit these settings for processing.*

Parameters

in	win	An MML_GDC_DISP_WINDOW returned from a previous call to mmlGdcDispWinCreate.
in	pname	Parameter name. See MML_GDC_DISP_WIN_ATTR for valid values.
in	param	Value to set for parameter pname.

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	If one of the parameters is invalid.

11.5.5.14 MM_ERROR mmlGdcDispWinSetBlendMode (MML_GDC_DISP_WINDOW win, MM_U32 blend_mode)

Sets blending mode.

Csrc: Source color

Asrc: Source alpha

Agbl: Global alpha

Aext: External alpha

Cdst: (Blend) destination color (alpha value of destination is not used)

Ctrans: Transparency color

Cout: Output color from this layer blend unit

As = 1;

```
if (((mode & MML_GDC_DISP_BLEND_TRANSPARENCY) ==
MML_GDC_DISP_BLEND_TRANSPARENCY) && (Ctrans == Csrc))
    As = 0;
if ((mode & MML_GDC_DISP_BLEND_GLOBAL_ALPHA) ==
MML_GDC_DISP_BLEND_GLOBAL_ALPHA)
    As = As * Agbl;
Ad = As;
if ((mode & MML_GDC_DISP_BLEND_SOURCE_ALPHA) ==
MML_GDC_DISP_BLEND_SOURCE_ALPHA)
    Ad = Ad * Asrc;
if((mode&MML_GDC_DISP_BLEND_SOURCE_MULTIPLY_ALPHA)==MML_GDC_DISP_BLEND_SOURC
E_MULTIPLY_ALPHA)
    As = As * Asrc;
Cout = Csrc * As + Cdst * (1 - Ad);
```

Note:

- *The blend mode settings of the window does not becomes active immediately with the related mmlGdcDispWinSetBlendMode call, but will be queued together with other settings of this window. Use mmlGdcDispWinCommit to submit these settings for processing.*
  Transparency is not supported for the YUV format.
  If blend mode MML_GDC_DISP_BLEND_TRANSPARENCY is selected, set transparency color by mmlGdcDispWinSetAttribute with attribute MML_GDC_DISP_WIN_ATTR_COLOR.
  If blend mode MML_GDC_DISP_BLEND_GLOBAL_ALPHA is selected, set global alpha ratio by mmlGdcDispWinSetAttribute with attribute MML_GDC_DISP_WIN_ATTR_COLOR.
  If the matrix set to the window (see mmlGdcDispWinSetMatrix()) is with a scaler factor, then only blend mode MML_GDC_DISP_BLEND_NONE and MML_GDC_DISP_BLEND_GLOBAL_ALPHA are allowed.

Parameters

in	win	An MML_GDC_DISP_WINDOW returned from a previous call to mmlGdcDispWinCreate.
in	blend_mode	Blend mode related parameter can be a bit field combination of: – MML_GDC_DISP_BLEND_NONE = Disable blending. – MML_GDC_DISP_BLEND_TRANSPARENCY = Enable transparency. – MML_GDC_DISP_BLEND_GLOBAL_ALPHA = Enable global alpha blending. – MML_GDC_DISP_BLEND_SOURCE_ALPHA = Enable per pixel source alpha blending. – MML_GDC_DISP_BLEND_SOURCE_MULTIPLY_ALPHA = Enable source alpha multiplication. – The color components RR, GG, BB are always 8 bit values also for 16 bpp and indexed color modes. For instance 0x00ffffff disable the 0xffff color entry in a 16 bpp buffer. The default blend mode is MML_GDC_DISP_BLEND_NONE.

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	If an invalid argument was passed.

11.5.5.15 MM_ERROR mmlGdcDispWinSetMatrix (MML_GDC_DISP_WINDOW win, MM_U32 target, const MM_FLOAT * matrix)

Set transformation matrix on window color buffer and/or the extern alpha buffer for scaling, rotation and flipping. The formula for the transformation based on this matrix is:

xout = matrix[0] * x + matrix[2] * y + matrix[4]

yout = matrix[1] * x + matrix[3] * y + matrix[5]

If matrix = NULL, following data is set in transformation matrix:

$$\begin{pmatrix} \text{matrix}[0] & \text{matrix}[2] & \text{matrix}[4] \\ \text{matrix}[1] & \text{matrix}[3] & \text{matrix}[5] \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Note:

- *The matrix settings of the window does not becomes active immediately with the related mmlGdcDispWinSetMatrix call, but will be queued together with other settings of this window. Use mmlGdcDispWinCommit to submit these settings for processing.*
  The allowed matrix properties differ depending on the window features. All windows support a panning matrix (surface move inside the layer) except if the feature MML_GDC_DISP_FEATURE_DECODE was requested.
  $$\begin{pmatrix} 1 & 0 & xoffset \\ 0 & 1 & yoffset \end{pmatrix}$$
  A mirror matrix can be used for windows without the above features and MML_GDC_DISP_FEATURE_MULTI_LAYER.
  $$\begin{pmatrix} -1 & 0 & xoffset \\ 0 & -1 & yoffset \end{pmatrix}$$
  Down scaling is not supported by display.
  Not all of rotation angles are supported by display. A rotation must be 0, 90, 180 or 270 degrees.
  The extern alpha buffer cannot be scaled.
  The buffer larger than window will be cut to fit the window size. The YUV buffer cannot be cut to odd pixel width.

Parameters

in	win	An MML_GDC_DISP_WINDOW returned from a previous call to mmlGdcDispWinCreate.
in	target	The target where the matrix is set to, the related parameter must be: – MML_GDC_DISP_BUFF_TARGET_COLOR_BUFF = Color buffer as target.
in	matrix	Transformation 3x2 matrix for scaling, rotation and flip.

Return values

MML_OK	On success. Otherwise the related error code.

11.5.5.16 MM_ERROR mmlGdcDispWinSetSurface (MML_GDC_DISP_WINDOW win, MM_U32 target, MML_GDC_SURFACE surf)

Show the surface content on a previously opened window.

Note:

- *The function will not be executed immediately but will be queued together with other modifications of this window. Use mmlGdcDispWinCommit to submit for processing.*
  If the surface describes an indexed color format, the driver will apply this color table to the hardware only if the window was created with the feature MML_GDC_DISP_FEATURE_INDEX_COLOR.

Parameters

in	win	An MML_GDC_DISP_WINDOW returned from a previous call to mmlGdcDispWinCreate.
in	target	The target where the surface is set to, must be: – MML_GDC_DISP_BUFF_TARGET_COLOR_BUFF = Color buffer as target.
in	surf	The MML_GDC_SURFACE object to show.

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	If an invalid argument was passed.

11.5.5.17 MM_ERROR mmlGdcDispWinSync (MML_GDC_DISP_WINDOW win, MML_GDC_SYNC sync)

Inserts a sync object into the window settings queue. The sync object will be signaled after the preceding mmlGdcDispWinCommit has been processed.

Parameters

in	win	A MML_GDC_DISP_WINDOW returned from a previous call to mmlGdcDispWinCreate.
out	sync	Sync object. After successful completion of mmlGdcDispWinSync it holds the parameter of the inserted sync.

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	If one of the parameters is invalid.

11.5.5.18 MM_ERROR mmlGdcDispWinWaitSync (MML_GDC_DISP_WINDOW win, MML_GDC_SYNC sync)

Inserts a sync wait into the window settings queue for win. mmlGdcDispWinCommit operations performed after this call are only executed after sync gets signaled. mmlGdcDispWinWaitSync shall only be called once before a call to mmlGdcDispWinCommit.

Parameters

in	win	The window to apply the wait condition.
in	sync	Sync object to wait for.

Return values

MML_OK	On success.
MML_ERR_GDC_DISP_INVALID_ARG	If one of the parameters is invalid.

11.6 Pixel Engine API

Pixel Engine (PixEng) API.

Data Structures

- struct MML_GDC_PE_CONTEXT_CONTAINER

Macros

- #define MML_GDC_PE_API extern
- #define MML_GDC_PE_STORE 0x00000001U
- #define MML_GDC_PE_SRC 0x00000002U
- #define MML_GDC_PE_DST 0x00000004U
- #define MML_GDC_PE_MASK 0x00000008U
- #define MML_GDC_PE_ROP_BLACKNESS ((MM_U08)0x00)
- #define MML_GDC_PE_ROP_WHITENESS ((MM_U08)0xFF)
- #define MML_GDC_PE_ROP_SRCCOPY ((MM_U08)0xAA)
- #define MML_GDC_PE_ROP_NOTSRCCOPY ((MM_U08)0x55)
- #define MML_GDC_PE_ROP_MASKCOPY ((MM_U08)0xCC)
- #define MML_GDC_PE_ROP_NOTMASK ((MM_U08)0x33)
- #define MML_GDC_PE_ROP_MASKINVERT ((MM_U08)0x66)
- #define MML_GDC_PE_ROP_MSKAND ((MM_U08)0x88)
- #define MML_GDC_PE_ROP_MASKERASE ((MM_U08)0x22)
- #define MML_GDC_PE_ROP_NOTMASKERASE ((MM_U08)0x11)
- #define MML_GDC_PE_ROP_MERGEMASK ((MM_U08)0xEE)
- #define MML_GDC_PE_ROP_MERGEMASKNOT ((MM_U08)0xBB)
- #define MML_GDC_PE_ROP_DSTCOPY ((MM_U08)0xF0)
- #define MML_GDC_PE_ROP_NOTDSTCOPY ((MM_U08)0x0F)
- #define MML_GDC_PE_ROP_DSTPAINT ((MM_U08)0xFE)
- #define MML_GDC_PE_ROP_MASKSEL ((MM_U08)0xB8)
- #define MML_GDC_PE_ROP_DSTAND ((MM_U08)0x80)
- #define MML_GDC_PE_FILTER_NEAREST 0
- #define MML_GDC_PE_FILTER_BILINEAR 1
- #define MML_GDC_PE_ATTR_ZERO_TOP_LEFT 0U
- #define MML_GDC_PE_ATTR_ZERO_BOTTOM_LEFT 1U
- #define MML_GDC_PE_TILE_FILL_ZERO 0U
- #define MML_GDC_PE_TILE_FILL_CONSTANT 1U
- #define MML_GDC_PE_TILE_PAD 2U
- #define MML_GDC_PE_TILE_PAD_ZERO 3U

Typedefs

- typedef MML_GDC_PE_CONTEXT_CONTAINER * MML_GDC_PE_CONTEXT

Enumerations

- enum MML_GDC_PE_CTX_ATTR {
  MML_GDC_PE_CTX_ATTR_DITHER_COLOR,
  MML_GDC_PE_CTX_ATTR_DITHER_ALPHA,
  MML_GDC_PE_CTX_ATTR_DITHER_OFFSET,
  MML_GDC_PE_CTX_ATTR_FILTER,
  MML_GDC_PE_ATTR_ZERO_POINT
  }

- enum MML_GDC_PE_SURF_ATTR {
    MML_GDC_PE_SURF_ATTR_COLORMULTI,
    MML_GDC_PE_SURF_ATTR_ALPHAMULTI,
    MML_GDC_PE_SURF_ATTR_TILE_MODE,
    MML_GDC_PE_SURF_ATTR_USE_CLIPPING
  }
- enum MML_GDC_PE_BF {
    MML_GDC_PE_BF_GL_ZERO = 0x0U,
    MML_GDC_PE_BF_GL_ONE = 0x1U,
    MML_GDC_PE_BF_GL_SRC_COLOR = 0x300U,
    MML_GDC_PE_BF_GL_ONE_MINUS_SRC_COLOR = 0x301U,
    MML_GDC_PE_BF_GL_SRC_ALPHA = 0x302U,
    MML_GDC_PE_BF_GL_ONE_MINUS_SRC_ALPHA = 0x303U,
    MML_GDC_PE_BF_GL_DST_ALPHA = 0x304U,
    MML_GDC_PE_BF_GL_ONE_MINUS_DST_ALPHA = 0x305U,
    MML_GDC_PE_BF_GL_DST_COLOR = 0x306U,
    MML_GDC_PE_BF_GL_ONE_MINUS_DST_COLOR = 0x307U,
    MML_GDC_PE_BF_GL_SRC_ALPHA_SATURATE = 0x308U,
    MML_GDC_PE_BF_GL_CONSTANT_COLOR = 0x8001U,
    MML_GDC_PE_BF_GL_ONE_MINUS_CONSTANT_COLOR = 0x8002U,
    MML_GDC_PE_BF_GL_CONSTANT_ALPHA = 0x8003U,
    MML_GDC_PE_BF_GL_ONE_MINUS_CONSTANT_ALPHA = 0x8004U
  }
- enum MML_GDC_PE_BM {
    MML_GDC_PE_BM_GL_FUNC_ADD = 0x8006U,
    MML_GDC_PE_BM_GL_MIN = 0x8007U,
    MML_GDC_PE_BM_GL_MAX = 0x8008U,
    MML_GDC_PE_BM_GL_FUNC_SUBTRACT = 0x800AU,
    MML_GDC_PE_BM_GL_FUNC_REVERSE_SUBTRACT = 0x800BU,
    MML_GDC_PE_BM_VG_BLEND_SRC = 0x2000U,
    MML_GDC_PE_BM_VG_BLEND_SRC_OVER = 0x2001U,
    MML_GDC_PE_BM_VG_BLEND_DST_OVER = 0x2002U,
    MML_GDC_PE_BM_VG_BLEND_SRC_IN = 0x2003U,
    MML_GDC_PE_BM_VG_BLEND_DST_IN = 0x2004U,
    MML_GDC_PE_BM_VG_BLEND_MULTIPLY = 0x2005U,
    MML_GDC_PE_BM_VG_BLEND_SCREEN = 0x2006U,
    MML_GDC_PE_BM_VG_BLEND_DARKEN = 0x2007U,
    MML_GDC_PE_BM_VG_BLEND_LIGHTEN = 0x2008U,
    MML_GDC_PE_BM_VG_BLEND_ADDITIVE = 0x2009U
  }
- enum MML_GDC_PE_CMATRIX_FORMAT {
    MML_GDC_PE_CMATRIX_FORMAT_4X3 = 0
  }
- enum MML_GDC_PE_GEO_MATRIX_FORMAT {
    MML_GDC_PE_GEO_MATRIX_FORMAT_3X2,
    MML_GDC_PE_GEO_MATRIX_FORMAT_3X3
  }
- enum MML_GDC_PE_CLUT_FORMAT {
    MML_GDC_PE_CLUT_FORMAT_33 = 33,
    MML_GDC_PE_CLUT_FORMAT_256 = 256
  }
- enum MML_GDC_PE_FILTER_CHANNEL {
    MML_GDC_PE_FILTER_CHANNEL_R = (1U<<3),

MML_GDC_PE_FILTER_CHANNEL_G = (1U<<2),
MML_GDC_PE_FILTER_CHANNEL_B = (1U<<1),
MML_GDC_PE_FILTER_CHANNEL_A = 1U,
MML_GDC_PE_FILTER_CHANNEL_RGB = (MML_GDC_PE_FILTER_CHANNEL_R | MML_GDC_PE_FILTER_CHANNEL_G | MML_GDC_PE_FILTER_CHANNEL_B),
MML_GDC_PE_FILTER_CHANNEL_RGBA= (MML_GDC_PE_FILTER_CHANNEL_R | MML_GDC_PE_FILTER_CHANNEL_G | MML_GDC_PE_FILTER_CHANNEL_B | MML_GDC_PE_FILTER_CHANNEL_A)
}

- enum MML_GDC_PE_FILTER_COLOR_FORMAT {
  MML_GDC_PE_FILTER_COLOR_FORMAT_R8G8B8,
  MML_GDC_PE_FILTER_COLOR_FORMAT_R5G6B5A8,
  MML_GDC_PE_FILTER_COLOR_FORMAT_R8G8B8A8,
  MML_GDC_PE_FILTER_COLOR_FORMAT_R10G10B10A8
  }

Functions

- MML_GDC_PE_API MM_ERROR mmlGdcPeResetContext (MML_GDC_PE_CONTEXT pectx)
- MML_GDC_PE_API MM_ERROR mmlGdcPeBindSurface (MML_GDC_PE_CONTEXT pectx, MM_U32 target, MML_GDC_SURFACE surface)
- MML_GDC_PE_API MM_ERROR mmlGdcPeAttribute (MML_GDC_PE_CONTEXT pectx, MML_GDC_PE_CTX_ATTR pname, MM_U32 param)
- MML_GDC_PE_API MM_ERROR mmlGdcPeColor (MML_GDC_PE_CONTEXT pectx, MM_U08 red, MM_U08 green, MM_U08 blue, MM_U08 alpha)
- MML_GDC_PE_API MM_ERROR mmlGdcPeSurfAttribute (MML_GDC_PE_CONTEXT pectx, MM_U32 target, MML_GDC_PE_SURF_ATTR pname, MM_U32 param)
- MML_GDC_PE_API MM_ERROR mmlGdcPeSurfColor (MML_GDC_PE_CONTEXT pectx, MM_U32 target, MM_U08 red, MM_U08 green, MM_U08 blue, MM_U08 alpha)
- MML_GDC_PE_API MM_ERROR mmlGdcPeBlendFunc (MML_GDC_PE_CONTEXT pectx, MML_GDC_PE_BF func_red_src, MML_GDC_PE_BF func_red_dst, MML_GDC_PE_BF func_green_src, MML_GDC_PE_BF func_green_dst, MML_GDC_PE_BF func_blue_src, MML_GDC_PE_BF func_blue_dst, MML_GDC_PE_BF func_alpha_src, MML_GDC_PE_BF func_alpha_dst)
- MML_GDC_PE_API MM_ERROR mmlGdcPeBlendMode (MML_GDC_PE_CONTEXT pectx, MML_GDC_PE_BM mode_red, MML_GDC_PE_BM mode_green, MML_GDC_PE_BM mode_blue, MML_GDC_PE_BM mode_alpha)
- MML_GDC_PE_API MM_ERROR mmlGdcPeRopOperation (MML_GDC_PE_CONTEXT pectx, MM_U08 op_red, MM_U08 op_green, MM_U08 op_blue, MM_U08 op_alpha)
- MML_GDC_PE_API MM_ERROR mmlGdcPeSetMatrix (MML_GDC_PE_CONTEXT pectx, MM_U32 target, MML_GDC_PE_GEO_MATRIX_FORMAT format, const MM_FLOAT ∗fMatrix)
- MML_GDC_PE_API MM_ERROR mmlGdcPeCLUTData (MML_GDC_PE_CONTEXT pectx, MML_GDC_PE_CLUT_FORMAT format, const MM_S16 ∗pRed, const MM_S16 ∗pGreen, const MM_S16 ∗pBlue)
- MML_GDC_PE_API MM_ERROR mmlGdcPeColorMatrix (MML_GDC_PE_CONTEXT pectx, MML_GDC_PE_CMATRIX_FORMAT format, const MM_FLOAT ∗fMatrix)
- MML_GDC_PE_API MM_ERROR mmlGdcPeGetDrawBox (MML_GDC_PE_CONTEXT pectx, MM_U32 ∗x, MM_U32 ∗y, MM_U32 ∗w, MM_U32 ∗h, MM_U32 reset)
- MML_GDC_PE_API MM_ERROR mmlGdcPeActiveArea (MML_GDC_PE_CONTEXT pectx, MM_U32 target, MM_S32 x, MM_S32 y, MM_U32 w, MM_U32 h)
- MML_GDC_PE_API MM_ERROR mmlGdcPeSelectArea (MML_GDC_PE_CONTEXT pectx, MM_U32 target)
- MML_GDC_PE_API MM_ERROR mmlGdcPeFill (MML_GDC_PE_CONTEXT pectx, MM_U32 x, MM_U32 y, MM_U32 w, MM_U32 h)

- MML_GDC_PE_API MM_ERROR mmlGdcPeBlt (MML_GDC_PE_CONTEXT pectx, MM_FLOAT offsetx, MM_FLOAT offsety)
- MML_GDC_PE_API MM_ERROR mmlGdcPeFinish (void)
- MML_GDC_PE_API MM_ERROR mmlGdcPeFlush (void)
- MML_GDC_PE_API MM_ERROR mmlGdcPeSync (MML_GDC_SYNC sync)
- MML_GDC_PE_API MM_ERROR mmlGdcPeWaitSync (MML_GDC_SYNC sync)
- MM_ERROR mmlGdcPeWaitForDispFrameEnd (MML_GDC_DISPLAY display, MM_U32 line)

11.6.1 Detailed Description

Pixel Engine (PixEng) API.

#include "mml_gdc_pixeng.h"

The pixel engine API provides all functions for blit operations using the pixel engine (2D core blit) hardware. As mentioned in the Overview Pixel Engine (PixEng), it requires MML_GDC_SURFACE objects to describe the pixel buffers and a MML_GDC_PE_CONTEXT object to describe the requested pixel operation.

The following example demonstrates the steps to fill a target buffer with blue and blend a src surface at the center of target buffer:

```
MML_GDC_PE_CONTEXT_CONTAINER ctx;
// reset the MML_GDC_PE_CONTEXT object
mmlGdcPeResetContext(&ctx);
// Bind a target surface as STORE and blend DeSTination buffer to the context.
// (The target pixel will be read, blended with src and written back to the target buffer.)
mmlGdcPeBindSurface(&ctx, MML_GDC_PE_STORE | MML_GDC_PE_DST,    target);
// Set a fill color
mmlGdcPeColor(&ctx, 0, 0, 255, 0);
// Fill the store buffer
mmlGdcPeFill(&ctx, 0, 0, target_width, target_height);
// Attache the source buffer
mmlGdcPeBindSurface(&ctx, MML_GDC_PE_SRC, src);

// Blend the source pixel to the target
mmlGdcPeBlt(&ctx, (target_width - src_width)/2, (target_height - src_height)/2);

// Optional: Ensure that the operation finished execution
mmlGdcPeFinish();
```

11.6.2 Macro Definition Documentation

11.6.2.1 #define MML_GDC_PE_API extern

Placeholder for export changes.

11.6.2.2 #define MML_GDC_PE_ATTR_ZERO_BOTTOM_LEFT 1U

The coordinate system for geometry operation starts in the lower left corner.

11.6.2.3 #define MML_GDC_PE_ATTR_ZERO_TOP_LEFT 0U

The coordinate system for geometry operation starts in the upper left corner.

Note:

- *For blit operation, it is equal to buffer content and display coordinate orientation. For draw operation, it means the buffer content orientation is mirrored.*

11.6.2.4 #define MML_GDC_PE_DST 0x00000004U

= Background for blend operations.

11.6.2.5 #define MML_GDC_PE_FILTER_BILINEAR 1

Bilinear filter enable.

11.6.2.6 #define MML_GDC_PE_FILTER_NEAREST 0

Nearest filter enable.

11.6.2.7 #define MML_GDC_PE_MASK 0x00000008U

= Mask surface.

11.6.2.8 #define MML_GDC_PE_ROP_BLACKNESS ((MM_U08)0x00)

= 0

11.6.2.9 #define MML_GDC_PE_ROP_DSTAND ((MM_U08)0x80)

= DST & MASK & SRC

11.6.2.10 #define MML_GDC_PE_ROP_DSTCOPY ((MM_U08)0xF0)

= DST

11.6.2.11 #define MML_GDC_PE_ROP_DSTPAINT ((MM_U08)0xFE)

= DST | MASK | SRC

11.6.2.12 #define MML_GDC_PE_ROP_MASKCOPY ((MM_U08)0xCC)

= MASK

11.6.2.13 #define MML_GDC_PE_ROP_MASKERASE ((MM_U08)0x22)

= SRC & ~MASK

11.6.2.14 #define MML_GDC_PE_ROP_MASKINVERT ((MM_U08)0x66)

= MASK ∧ SRC

11.6.2.15 #define MML_GDC_PE_ROP_MASKSEL ((MM_U08)0xB8)

= MASK ? SRC : DST

11.6.2.16 #define MML_GDC_PE_ROP_MERGEMASK ((MM_U08)0xEE)

= SRC | MASK

11.6.2.17 #define MML_GDC_PE_ROP_MERGEMASKNOT ((MM_U08)0xBB)

= SRC | ~MASK

11.6.2.18 #define MML_GDC_PE_ROP_MSKAND ((MM_U08)0x88)

= MASK & SRC

11.6.2.19 #define MML_GDC_PE_ROP_NOTDSTCOPY ((MM_U08)0x0F)

= ~DST

11.6.2.20 #define MML_GDC_PE_ROP_NOTMASK ((MM_U08)0x33)

= ~MASK

11.6.2.21 #define MML_GDC_PE_ROP_NOTMASKERASE ((MM_U08)0x11)

= ~ (MASK | SRC)

11.6.2.22 #define MML_GDC_PE_ROP_NOTSRCCOPY ((MM_U08)0x55)

= ~SRC

11.6.2.23 #define MML_GDC_PE_ROP_SRCCOPY ((MM_U08)0xAA)

= SRC

11.6.2.24 #define MML_GDC_PE_ROP_WHITENESS ((MM_U08)0xFF)

= 1

11.6.2.25 #define MML_GDC_PE_SRC 0x00000002U

= blit source surface.

11.6.2.26 #define MML_GDC_PE_STORE 0x00000001U

= blit write target.

11.6.2.27 #define MML_GDC_PE_TILE_FILL_CONSTANT 1U

Samples outside the frame are filled with constant color.

11.6.2.28 #define MML_GDC_PE_TILE_FILL_ZERO 0U

Samples outside the frame are treated as zero pixel value.

11.6.2.29 #define MML_GDC_PE_TILE_PAD 2U

Samples outside the frame are padded with the last valid border pixels.

11.6.2.30 #define MML_GDC_PE_TILE_PAD_ZERO 3U

Applies tile mode PAD to RGB channels and tile mode ZERO to alpha channel.

11.6.3 Typedef Documentation

11.6.3.1 typedef MML_GDC_PE_CONTEXT_CONTAINER* MML_GDC_PE_CONTEXT

The pixel engine context object definition.

11.6.4 Enumeration Type Documentation

11.6.4.1 enum MML_GDC_PE_BF

Blit Blend function definition used by mmlGdcPeBlendFunc.

11.6.4.2 enum MML_GDC_PE_BM

Blit Blend mode definition used by mmlGdcPeBlendMode.

11.6.4.3 enum MML_GDC_PE_CLUT_FORMAT

CLUT entities size.

Enumerator

MML_GDC_PE_CLUT_FORMAT_33
Each array for RGB contains 33 10-bit values to describe the 0-255 index range. The missing values are interpolated (see mmlGdcPeCLUTData for details).

MML_GDC_PE_CLUT_FORMAT_256
Each array for RGB contains 256 values to describe the CLUT.

11.6.4.4 enum MML_GDC_PE_CMATRIX_FORMAT

Color matrix format.

Enumerator

MML_GDC_PE_CMATRIX_FORMAT_4X3
float[12] array with 4 column and 3 lines.

11.6.4.5 enum MML_GDC_PE_CTX_ATTR

Context attributes used by mmlGdcPeAttribute.

Enumerator

MML_GDC_PE_CTX_ATTR_DITHER_COLOR
Set the color dither mode. The related parameter can be

- MM_TRUE Enable color dithering.
- MM_FALSE Disable color dithering (default).

MML_GDC_PE_CTX_ATTR_DITHER_ALPHA
Set the alpha dither mode. The related parameter can be

- MM_TRUE Enable alpha dithering.
- MM_FALSE Disable alpha dithering (default).

MML_GDC_PE_CTX_ATTR_DITHER_OFFSET
Set a dither offset. The value can be in the range from (0..15).

Note:
- *If the application renders a new frame with the vsync display frame rate, the quality may be improved by increasing this dither offset value with each frame. It is the same effect like dithering a 8-bit color buffer component to a 6 bit panel. Using this feature it is possible to use a smaller render buffer with the same color quality.*

Warning:
- *For target buffers with few bits (e.g., <=4) per color component it will cause visible flickering artifacts.*

MML_GDC_PE_CTX_ATTR_FILTER
Set the filter mode. The related parameter can be

- MML_GDC_PE_FILTER_NEAREST.
- MML_GDC_PE_FILTER_BILINEAR (default).
- ::MML_GDC_PE_FILTER_ANISOTROPIC.

MML_GDC_PE_ATTR_ZERO_POINT
Define the coordinate zero point for geometry operations. See also Coordinate System Hints. The related parameter can be

- MML_GDC_PE_ATTR_ZERO_TOP_LEFT.
- MML_GDC_PE_ATTR_ZERO_BOTTOM_LEFT (default).

11.6.4.6 enum MML_GDC_PE_FILTER_CHANNEL

Color channels for filter.

Enumerator

MML_GDC_PE_FILTER_CHANNEL_R
Filter is applied to R or Y channel.

MML_GDC_PE_FILTER_CHANNEL_G
Filter is applied to G or U channel.

MML_GDC_PE_FILTER_CHANNEL_B
Filter is applied to B or V channel.

MML_GDC_PE_FILTER_CHANNEL_A
Filter is applied to Alpha channel (not available for MML_GDC_PE_FILTER_COLOR_FORMAT_R8G8B8).

MML_GDC_PE_FILTER_CHANNEL_RGB
Filter is applied to RGB or YUV channel

MML_GDC_PE_FILTER_CHANNEL_RGBA
Filter is applied to RGBA or YUVA channel (not available for MML_GDC_PE_FILTER_COLOR_FORMAT_R8G8B8).

11.6.4.7 enum MML_GDC_PE_FILTER_COLOR_FORMAT

Filter color formates.

Enumerator

MML_GDC_PE_FILTER_COLOR_FORMAT_R8G8B8
RGB888 format. Alpha is not filtered but set to constant value 255.

MML_GDC_PE_FILTER_COLOR_FORMAT_R5G6B5A8
RGBA5658 format.

MML_GDC_PE_FILTER_COLOR_FORMAT_R8G8B8A8
RGBA8888 format.

Note:
- *Only available for MML_GDC_PE_FILTER_TYPE_FIR5X4.*

MML_GDC_PE_FILTER_COLOR_FORMAT_R10G10B10A8
RGBA1010108 format.

Note:
- *Only available for MML_GDC_PE_FILTER_TYPE_FIR5X3.*

11.6.4.8 enum MML_GDC_PE_GEO_MATRIX_FORMAT

Geometry matrix format.

Enumerator

MML_GDC_PE_GEO_MATRIX_FORMAT_3X2
float[6] array with 3 column and 2 lines.

MML_GDC_PE_GEO_MATRIX_FORMAT_3X3
float[9] array with 3 column and 3 lines.

11.6.4.9 enum MML_GDC_PE_SURF_ATTR

Surface attributes used by mmlGdcPeSurfAttribute.

Enumerator

MML_GDC_PE_SURF_ATTR_COLORMULTI
Enable/disable of color multiplication. The related parameter can be

- MM_TRUE Enable color multiplication.
- MM_FALSE Disable color multiplication (default). The related formula is

```
if (ColorMultiply == MM_TRUE)
    Cout = Cin * Aout; // (Aout see MML_GDC_PE_SURF_ATTR_ALPHAMULTI)
else
```

```
    Cout = Cin;
```

MML_GDC_PE_SURF_ATTR_ALPHAMULTI

Enable/disable multiplication of pixel alpha with constant alpha defined by mmlGdcPeSurfColor(). The related parameter can be

- MM_TRUE Enable alpha multiplication.
- MM_FALSE Disable alpha multiplication (default). The related formula is

```
if (AlphaMultiply == MM_TRUE)
    Aout = Ain * Aconst;
else
    Aout = Ain;
```

MML_GDC_PE_SURF_ATTR_TILE_MODE

Mode of tiling mode for pixels outside of source buffer. The related parameter can be

- MML_GDC_PE_TILE_FILL_ZERO.
- MML_GDC_PE_TILE_FILL_CONSTANT.
- MML_GDC_PE_TILE_PAD.
- MML_GDC_PE_TILE_PAD_ZERO (default).

Note:

- *Compressed and YUV422 images can only be used with MML_GDC_PE_TILE_FILL_ZERO. The MML_GDC_PE_SURF_ATTR_TILE_MODE settings will be ignored for such images.*

MML_GDC_PE_SURF_ATTR_USE_CLIPPING

Define whether or not the surface coordinates given by mmlGdcPeActiveArea are used as clip coordinates while reading (SRC, DST, MASK) or writing (STORE) the surface. If USE_CLIPPING is disabled the ActiveArea coordinates are used for the target blit bounding box calculation only. If USE_CLIPPING is enabled the surface will be used like a smaller bitmap.

Note:

While using clipping for source surfaces, the attribute MML_GDC_PE_SURF_ATTR_TILE_MODE must set to MML_GDC_PE_TILE_FILL_ZERO.

The bounding box defined by mmlGdcPeActiveArea() will be always used as clipping box if USE_CLIPPING is enabled. (Independent of the mmlGdcPeSelectArea() settings.)

- *MM_FALSE (default): disable CLIP feature.*
- *MM_TRUE: enable clip feature.*

11.6.5 Function Documentation

11.6.5.1 MML_GDC_PE_API MM_ERROR mmlGdcPeActiveArea(MML_GDC_PE_CONTEXT pectx, MM_U32 target, MM_S32 x, MM_S32 y, MM_U32 w, MM_U32 h)

mmlGdcPeActiveArea defines the processing area for the surface that is bound to the specified target. See also mmlGdcPeSelectArea and MML_GDC_PE_SURF_ATTR_USE_CLIPPING.

The area is defined by lower left coordinate, width and height. The lower left coordinate is inside of processing area. The upper right coordinate (x+w, y+h) is outside of processing area.

Parameters must be x < x+w and y < y+h. If x or y is equal to 4096, function returns MML_ERR_GDC_PE_INVALID_PARAMETER. If w and h are equal to 0, active area is disabled. If x and y are negative, the color value is defined by mmlGdcPeSurfAttribute and MML_GDC_PE_SURF_ATTR_TILE_MODE. If pectx is equal to NULL, mmlGdcPeActiveArea is terminated without any operation.

Note:

- *Blit operations with a non default mmlGdcPeActiveArea setting may fail and report an error if buffers are involved with a bit per pixel size different to multiple of 8bit or YUV color format.*

Parameters

in,out	pectx	Pixel Engine context (!=NULL).
in	target	[in] Setting target. It is a single or OR combined value of: MML_GDC_PE_SRC MML_GDC_PE_DST MML_GDC_PE_STORE MML_GDC_PE_MASK
in	x	Left start coordinate of the active area (-4095 - 4096).
in	y	Lower (or upper see MML_GDC_PE_ATTR_ZERO_POINT) start coordinate of the active area (-4095 - 4096).
in	w	Width of active area (0 - 4096).
in	h	Height of active area (0 - 4096).

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.2 MML_GDC_PE_API MM_ERROR mmlGdcPeAttribute(MML_GDC_PE_CONTEXT pectx, MML_GDC_PE_CTX_ATTR pname, MM_U32 param)

Set an attribute for the specified context.

If pectx is equal to NULL, mmlGdcPeAttribute is terminated without any operation.

Parameters

in,out	pectx	Pixel Engine context (!=NULL).
in	pname	State name for setting. Can be one of MML_GDC_PE_CTX_ATTR
in	param	Parameter for argument target (See MML_GDC_PE_CTX_ATTR description).

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.3 MML_GDC_PE_API MM_ERROR mmlGdcPeBindSurface (MML_GDC_PE_CONTEXT pectx, MM_U32 target, MML_GDC_SURFACE surface)

mmlGdcPeBindSurface is setting function for parameters about source, destination, mask and store surface.
If pectx is equal to NULL, mmlGdcPeBindSurface is terminated without any operation.

Note:

- *All bound surfaces must not be deleted as long as the context is used. Parameter changes in the surface object after binding are used for further blit operations with the context.*

Parameters

in,out	pectx	Pixel Engine context (!=NULL).
in	target	Binding target. It is a single or OR combined value of: MML_GDC_PE_SRC MML_GDC_PE_DST MML_GDC_PE_STORE MML_GDC_PE_MASK
in	surface	Surface object; NULL: unbind surface.

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.4 MML_GDC_PE_API MM_ERROR mmlGdcPeBlendFunc (MML_GDC_PE_CONTEXT pectx, MML_GDC_PE_BF func_red_src, MML_GDC_PE_BF func_red_dst, MML_GDC_PE_BF func_green_src, MML_GDC_PE_BF func_green_dst, MML_GDC_PE_BF func_blue_src, MML_GDC_PE_BF func_blue_dst, MML_GDC_PE_BF func_alpha_src, MML_GDC_PE_BF func_alpha_dst)

Set the blending parameter. If pectx is equal to NULL, mmlGdcPeBlendFunc is terminated without any operation. The following table shows the possible blend functions

- F stands for the selected blend function. See mmlGdcPeBlendMode for further usage.
- Cs, Cd represent the incoming color or alpha component.
- As, Ad represent the incoming alpha component.
- Cc, Ac represent the constant color or alpha component defined by mmlGdcPeColor.

Note:

- *The incoming color components Cs, Cd, As and Ad can be the original image color or a result of a previous operation. See MML_GDC_PE_SURF_ATTR_ALPHAMULTI and MML_GDC_PE_SURF_ATTR_COLORMULTI.*

Blend Function	RGBA Components
MML_GDC_PE_BF_GL_ZERO	F = 0
MML_GDC_PE_BF_GL_ONE	F = 1
MML_GDC_PE_BF_GL_SRC_COLOR	F = Cs
MML_GDC_PE_BF_GL_ONE_MINUS_SRC_COLOR	F = 1 - Cs
MML_GDC_PE_BF_GL_SRC_ALPHA	F = As
MML_GDC_PE_BF_GL_ONE_MINUS_SRC_ALPHA	F = 1 - As
MML_GDC_PE_BF_GL_DST_ALPHA	F = Ad
MML_GDC_PE_BF_GL_ONE_MINUS_DST_ALPHA	F = 1 - Ad
MML_GDC_PE_BF_GL_DST_COLOR	F = Cd
MML_GDC_PE_BF_GL_ONE_MINUS_DST_COLOR	F = 1 - Cd
MML_GDC_PE_BF_GL_SRC_ALPHA_SATURATE	F = min(As, 1 - Ad)
MML_GDC_PE_BF_GL_CONSTANT_COLOR	F = Cc
MML_GDC_PE_BF_GL_ONE_MINUS_CONSTANT_COLOR	F = 1 - Cc
MML_GDC_PE_BF_GL_CONSTANT_ALPHA	F = Ac
MML_GDC_PE_BF_GL_ONE_MINUS_CONSTANT_ALPHA	F = 1 - Ac

Note:

- *If OpenVG blend mode (See mmlGdcPeBlendMode) is used, setting for this function is ignored in drawing image.*

Parameters

in,out	pectx	Pixel Engine context (!=NULL).
in	func_red_src	Blend function of source red (default: MML_GDC_PE_BF_GL_SRC_ALPHA).
in	func_red_dst	Blend function of destination red (default: MML_GDC_PE_BF_GL_ONE_MINUS_SRC_ALPHA).
in	func_green_src	Blend function of source green (default: MML_GDC_PE_BF_GL_SRC_ALPHA).
in	func_green_dst	Blend function of destination green (default: MML_GDC_PE_BF_GL_ONE_MINUS_SRC_ALPHA).
in	func_blue_src	Blend function of source blue (default: MML_GDC_PE_BF_GL_SRC_ALPHA).
in	func_blue_dst	Blend function of destination blue (default: MML_GDC_PE_BF_GL_ONE_MINUS_SRC_ALPHA).
in	func_alpha_src	Blend function of source alpha (default: MML_GDC_PE_BF_GL_ONE).
in	func_alpha_dst	Blend function of destination alpha (default: MML_GDC_PE_BF_GL_ONE_MINUS_SRC_ALPHA).

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.5 MML_GDC_PE_API MM_ERROR mmlGdcPeBlendMode (MML_GDC_PE_CONTEXT pectx, MML_GDC_PE_BM mode_red, MML_GDC_PE_BM mode_green, MML_GDC_PE_BM mode_blue, MML_GDC_PE_BM mode_alpha)

Set the blending parameter. If pectx is equal to NULL, mmlGdcPeBlendMode is terminated without any operation.

Note:

- *The output of a blend operation is always alpha pre-multiplied. For the detail blend function, refer to chapter 13.2 in OpenVG specification Version 1.1 (March 27, 2007).*

The following table is a brief description of the different blend modes.

- Cs, Cd and C represents the incoming source, blend destination and result component: red, green, blue or alpha.
- As and Ad stands for incoming source and blend destination alpha component.
- Fs and Fd stands for incoming source and blend destination blend function. See mmlGdcPeBlendFunc.1

Note:

- *The incoming color components Cs, Cd, As and Ad can be the original image color or a result of a previous operation. See MML_GDC_PE_SURF_ATTR_ALPHAMULTI and MML_GDC_PE_SURF_ATTR_COLORMULTI.*

Blend Mode	RGBA Components
MML_GDC_PE_BM_GL_FUNC_ADD	C = Cs ∗ Fs + Cd ∗ Fd
MML_GDC_PE_BM_GL_MIN	C = min(Cs, Cd)
MML_GDC_PE_BM_GL_MAX	C = max(Cs, Cd)
MML_GDC_PE_BM_GL_FUNC_SUBTRACT	C = Cs ∗ Fs - Cd ∗ Fd
MML_GDC_PE_BM_GL_FUNC_REVERSE_SUBTRACT	C = Cd ∗ Fd - Cs ∗ Fs
MML_GDC_PE_BM_VG_BLEND_SRC	C = Cs
MML_GDC_PE_BM_VG_BLEND_SRC_OVER	C = Cs + Cd ∗ (1 - As)
MML_GDC_PE_BM_VG_BLEND_DST_OVER	C = Cs ∗ (1 - Ad)+ Cd
MML_GDC_PE_BM_VG_BLEND_SRC_IN	C = Cs ∗ Ad
MML_GDC_PE_BM_VG_BLEND_DST_IN	C = Cd ∗ As
MML_GDC_PE_BM_VG_BLEND_MULTIPLY	C = Cs ∗ (1-Ad) + Cd ∗ (1-As) + Cs ∗ Cd
MML_GDC_PE_BM_VG_BLEND_SCREEN	C = Cs + Cd - Cs ∗ Cd
MML_GDC_PE_BM_VG_BLEND_DARKEN	C = min(Cs + Cd ∗ (1-As), Cd + Cs ∗ (1-Ad))
MML_GDC_PE_BM_VG_BLEND_LIGHTEN	C = max(Cs + Cd ∗ (1-As), Cd + Cs ∗ (1-Ad))
MML_GDC_PE_BM_VG_BLEND_ADDITIVE	C = Cs + Cd

Parameters

in,out	pectx	Pixel Engine context (!=NULL).
in	mode_red	Blend mode of red (default: MML_GDC_PE_BM_GL_FUNC_ADD).
in	mode_green	Blend mode of green (default: MML_GDC_PE_BM_GL_FUNC_ADD).
in	mode_blue	Blend mode of blue (default: MML_GDC_PE_BM_GL_FUNC_ADD).
in	mode_alpha	Blend mode of alpha (default: MML_GDC_PE_BM_GL_FUNC_ADD).

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.6 MML_GDC_PE_API MM_ERROR mmlGdcPeBlt (MML_GDC_PE_CONTEXT pectx, MM_FLOAT offsetx, MM_FLOAT offsety)

This API initiates an operation that reads pixel data from surfaces bound to SRC, DST and MASK and performs a calculation using it. The resulting pixel data build a rectangle that is written to the bound STORE surface. The details of the operation are defined by the context and surface attributes.

Note

- *The offsetx and offsety position parameters will be added to the current geometric matrix of the src and mask surface. That means they are not really required because the matrix changes can handle the same. However the most common use case is to blend a (modified) source bitmap to a defined x, y position and it is much simpler to commit this position as parameter. The geometrical relation between pixels of the target buffer and pixels of the source buffer are defined in the following way:*
  - *Moffs represent a matrix using the fX, fY offsets given from this function.*
  - *Ms (Xs, Ys) represent the surface matrix (pixel) of the related source: SRC or MASK.*

$$\begin{pmatrix} \text{Xstore} \\ \text{Ystore} \end{pmatrix} = \text{Moffs} \times \text{Ms} \times \begin{pmatrix} \text{Xs} \\ \text{Ys} \end{pmatrix}$$

The path for the DST calculation is a little bit different:

$$\begin{pmatrix}\text{Xstore}\\ \text{Ystore}\end{pmatrix} = \text{Mdst} \times \begin{pmatrix}\text{Xs}\\ \text{Ys}\end{pmatrix}$$

- *A typical mmlGdcPeBlt operation processes a store rectangle defined by the active area of the SRC surface and the given matrix transformation. An application can change this behavior by using mmlGdcPeSelectArea.*
- *A SRC and STORE surface must be defined in minimum to proceed a mmlGdcPeBlt (simple copy) operation. If a DST surface is defined, a blend operation will be performed. If a MASK surface is defined the MASK alpha channel will be used as external alpha. That means the resulting alpha for the blending step is A = Asrc ∗Amask. If a ROP operation with MASK or DST is defined, external alpha or blending mutates to a ROP operation. See mmlGdcPeRopOperation for more details.*
- *The graphical operation will not be finished after the mmlGdcPeBlt call. That means the involved buffers are still in use. Please use synchronization objects or simple mmlGdcPeFinish to ensure that all operations are complete.*
- *Pixel Engine operations can be queued by the driver to enhance performance especially in multi-threading environment. The execution especially of long processing commands can be forced by a mmlGdcPeFlush call. mmlGdcPeFinish, mmlGdcDispWinSetSurface and mmlGdcPeSync also flush the command queue.*
- *The following features can be defined for the bounded surfaces:*
  - *ALL: simple transformation (translation, mirroring, 90°rotation) if buffer is not compressed.*
  - *SRC: rotate/scale or index/decompress.*
  - *DST: index/decompress if SRC does not require these features.*
  - *MASK: scale if scale factor is identical with SRC.*

Parameters

in,out	pectx	Pixel Engine context (!=NULL).
in	offsetx	Horizontal offset (-4096 - 4095).
in	offsety	Vertical offset (-4096 - 4095) (count direction depends on MML_GDC_PE_ATTR_ZERO_POINT).

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.7 MML_GDC_PE_API MM_ERROR mmlGdcPeCLUTData (MML_GDC_PE_CONTEXT pectx, MML_GDC_PE_CLUT_FORMAT format, const MM_S16 ∗ pRed, const MM_S16 ∗ pGreen, const MM_S16 pBlue)

Used to configure the Color Lookup Table (CLUT) (e.g., for gamma correction).

The format MML_GDC_PE_CLUT_FORMAT_256 defines 256 sample points representing the the resulting color channel intensity.

The format MML_GDC_PE_CLUT_FORMAT_33 defines 33 sample points representing the resulting color channel intensity. Intermediate values will be interpolated by the HW. The 1st sample point corresponds to input color code 0, 2nd one to 32, ..., last one to 1024 of the 10 bit 2D core internal processing pipeline. Although input 1024 is not possible, the last sample point is needed for interpolation of codes 993 to 1023.

An index entry of 0 stands for the minimum and 1023 for the maximum intensity. Index values outside this range will be clamped.

Note

Example: Let F(in) be the requested gamma formula. Input values of F(in) are in the range [0.0, 1.0]. It is allowed that the output value is smaller than 0.0 or bigger 1.0. The value array (in this example pRed) must be calculated in the following way:

- *MML_GDC_PE_CLUT_FORMAT_256:*

  ```
  for (i = 0; i <= 255; i++)
      pRed[i] = (MM_S16)(0.5f + ( F(i/255.0f) * 1023.0f));
  ```

- *MML_GDC_PE_CLUT_FORMAT_33:*

  ```
  for (i = 0; i <= 32; i++)
      pRed[i] = (MM_S16)(0.5f + ( F(i/32.0f * 1024.0f/1023.0f) * 1023));
  ```

  Please note that the given formula calculates the value for F(256/255). If F(x) is only defined for input values 0.0..1.0 then pRed[32] can be calculated as

  ```
  pRed[32] = (MM_S16)(0.5f + (( 32.0f * F(1) - F(31.0f * 32.0f / 1023.0f)) * 1023.0f / 31.0f));
  ```

The pRed, pGreen and pBlue pointers must be valid for all following mmlGdcPeBlt() calls.
If valid CLUT data is loaded, context attribute MML_GDC_PE_CTX_ATTR_GAMMA is set to MML_GDC_PE_GAMMA_NEUTRAL.
If one pointer of color components is NULL, then the CLUT is set to bypass.
If pectx is equal to NULL, mmlGdcPeCLUTData is terminated without any operation.

Parameters

in,out	pectx	Pixel Engine context (!=NULL).
in	format	Defines the number of entries in the array. Depending on the hardware the CLUT hardware may support not all format types. In this case the driver interpolates the missing or skips the needless values.
in	pRed	Pointer to array of red values. The size of the array depends on format.
in	pGreen	Pointer to array of green values. The size of the array depends on format.
in	pBlue	Pointer to array of blue values. The size of the array depends on format.

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.8 MML_GDC_PE_API MM_ERROR mmlGdcPeColor (MML_GDC_PE_CONTEXT pectx, MM_U08 red, MM_U08 green, MM_U08 blue, MM_U08 alpha)

Set the constant color value. This color has the following usage:

- Fill color used in mmlGdcPeFill.
- Constant color used in blend mode (for detail information, refer to mmlGdcPeBlendMode) If pectx is equal to NULL, mmlGdcPeColor is terminated without any operation.

Parameters

in,out	pectx	Pixel Engine context (!=NULL).
in	red	Red component of color (0 - 255, default 0).
in	green	Green component of color (0 - 255, default 0).
in	blue	Blue component of color (0 - 255, default 0).
in	alpha	Alpha component of color (0 - 255, default 0).

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.9 MML_GDC_PE_API MM_ERROR mmlGdcPeColorMatrix (MML_GDC_PE_CONTEXT pectx, MML_GDC_PE_CMATRIX_FORMAT format, const MM_FLOAT * fMatrix)

mmlGdcPeColorMatrix is setting function for color matrix. If pectx is equal to NULL, mmlGdcPeColorMatrix is terminated without any operation.
fMatrix is a 4x3 matrix (represented as float[12] array) for RGB modification.

red_out = fMatrix[0] * red + fMatrix[3] * green + fMatrix[6] * blue + fMatrix[9] * 255
green_out = fMatrix[1] * red + fMatrix[4] * green + fMatrix[7] * blue + fMatrix[10] * 255
blue_out = fMatrix[2] * red + fMatrix[5] * green + fMatrix[8] * blue + fMatrix[11] * 255
alpha_out = alpha

If fMatrix = NULL (default) the color matrix function will be switched off.

Note

- *If a color matrix is set using mmlGdcPeColorMatrix(), then driver internal automatic YUV to RGB conversion of SRC buffer will be shut off. The YUV color will be converted according to the user defined color matrix. The range for the multiplication factors is -3.5 .. 3.5. The range for the constant factors is -3.0 .. 3.0*

Parameters

in,out	pectx	Pixel Engine context (!=NULL).
in	format	Format of the matrix (must be MML_GDC_PE_CMATRIX_FORMAT_4X3).
in	fMatrix	Address of color matrix (See [Description])

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.10 MML_GDC_PE_API MM_ERROR mmlGdcPeFill (MML_GDC_PE_CONTEXT pectx, MM_U32 x, MM_U32 y, MM_U32 w, MM_U32 h)

This API fills the specified region of the surface that is bound to the MML_GDC_PE_STORE target with the constant color (see mmlGdcPeColor). If w or h is equal to 0, this API returns MM_TRUE but no work is done.

Note

- *The graphical operation will not be finished after the mmlGdcPeFill call. That means the target buffer may be still in use. Please use synchronization objects or simple mmlGdcPeFinish to ensure that all operations are complete if the buffer is used by another hardware unit (e.g., CPU, display) beside PixEng afterwards.*

Parameters

in	pectx	Pixel Engine context (!=NULL).
in	x	Left start coordinate of the store surface (0 - 4095).
in	y	Lower (or upper see MML_GDC_PE_ATTR_ZERO_POINT) start coordinate of the store surface (0 - 4095).
in	w	Width of rectangle region in pixel count (0 - 4096).

in	h	Height of rectangle region in line count (0 - 4096).

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.11 MML_GDC_PE_API MM_ERROR mmlGdcPeFinish (void)

This API is used to wait on blitting and drawing completion for synchronization.

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.12 MML_GDC_PE_API MM_ERROR mmlGdcPeFlush (void)

Force execution of PixEng commands in finite time.

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.13 MML_GDC_PE_API MM_ERROR mmlGdcPeGetDrawBox (MML_GDC_PE_CONTEXT pectx, MM_U32 *x, MM_U32 * y, MM_U32 * w, MM_U32 * h, MM_U32 reset)

mmlGdcPeGetDrawBox is a function to get the last draw box. Each Blt function calculates a store surface, draw rectangle basing on the mmlGdcPeSelectArea settings and the related surface properties (active area, matrix). The bounding box of this rectangle and the previously stored draw box will be stored as the new draw box. The draw box will be cleared if the reset parameter of mmlGdcPeGetDrawBox is different from 0. An application can use the draw box to get the minimal rectangle of a (frame) buffer that must be restored. The function returns an error if no blit operation was executed since the last reset.

Note

- *The draw box calculation based only on bounding box calculations for SRC, DST and MASK. Possible STORE settings does not influence the calculation.*
  The draw box is not influenced by Fill operations.

Parameters

in,out	pectx	Pixel Engine context (!=NULL).
in,out	x	Pointer to get horizontal start point.
in,out	y	Pointer to get vertical start point (zero point depends on MML_GDC_PE_ATTR_ZERO_POINT).
in,out	w	Pointer to get width.
in,out	h	Pointer to get height.
in	reset	Reset flag (see above).

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.14 MML_GDC_PE_API MM_ERROR mmlGdcPeResetContext (MML_GDC_PE_CONTEXT pectx)

Reset all parameters of the context object.

Parameters

in,out	pectx	Pixel Engine context.

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.15 MML_GDC_PE_API MM_ERROR mmlGdcPeRopOperation (MML_GDC_PE_CONTEXT pectx, MM_U08 op_red, MM_U08 op_green, MM_U08 op_blue, MM_U08 op_alpha)

Set the Raster Operation (ROP) for each color channel and the alpha channel. If pectx is equal to NULL, mmlGdcPeRopOperation is terminated without any operation.

Note

- *The involved source surfaces depend of the ROP mode. The driver will report an error if a requested surface is not defined and mmlGdcPeBlt is called.*
- *If one of the ROP modes uses the DST surface, the blend unit in the blit path will be switched off and the result will be written directly in the store surface.*
- *If there is a MASK surface, by default MASK buffer alpha channel is read as extern alpha value of SRC surface. If one of the ROP modes uses the MASK surface the extern alpha path of the SRC surface will be switched off and the MASK surface is the input of ROP operation.*
- *The required ROP mode can be calculated by the following table:*

surface	DST	MASK	SRC	output (STORE)
	0	0	0	0
	0	0	1	1
	0	1	0	0
	0	1	1	1
	1	0	0	1
	1	0	1	0
	1	1	0	1
	1	1	1	1
	Operation index			0x5B

Some useful ROP modes are predefined in the define section of this file, see MML_GDC_PE_ROP_...

Parameters

in,out	pectx	Pixel Engine context (!=NULL).
in	op_red	ROP3 operation code for red component (default: MML_GDC_PE_ROP_SRCCOPY).
in	op_green	ROP3 operation code for green component (default: MML_GDC_PE_ROP_SRCCOPY).
in	op_blue	ROP3 operation code for blue component (default: MML_GDC_PE_ROP_SRCCOPY).

in	op_alpha	ROP3 operation code for alpha component (default: MML_GDC_PE_ROP_SRCCOPY).

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.16 MML_GDC_PE_API MM_ERROR mmlGdcPeSelectArea (MML_GDC_PE_CONTEXT pectx, MM_U32 target)

mmlGdcPeSelectArea defines which surfaces are used to calculate the processing area. A default mmlGdcPeBlt function processes rectangle in store surface defined by the active area (see mmlGdcPeActiveArea) of the src surface and the given matrix transformation defined by mmlGdcPeSetMatrix. mmlGdcPeSelectArea changed it to active area of any other bounded surfaces or a combination of surfaces. Combination can be defined like this:

mmlGdcPeSelectArea(pectx, MML_GDC_PE_SRC|GDC_PE_DST); If more than one surface defines to target, the bounding box of all active areas will be used.

Parameters

in,out	pectx	Pixel Engine context (!=NULL)
in	target	[in] Selecting target. It is a single or OR combined value of:MML_GDC_PE_SRC (default) MML_GDC_PE_DST MML_GDC_PE_STORE MML_GDC_PE_MASK

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.17 MML_GDC_PE_API MM_ERROR mmlGdcPeSetMatrix (MML_GDC_PE_CONTEXT pectx, MM_U32 target, MML_GDC_PE_GEO_MATRIX_FORMAT format, const MM_FLOAT * fMatrix)

mmlGdcPeSetMatrix is setting function of transformation matrix for scaling, rotation, translation and flipping for all source surfaces: MML_GDC_PE_SRC, MML_GDC_PE_DST and MML_GDC_PE_MASK. The formula for the transformation based on this matrix is as follows:

xout = fMatrix[0] * xin + fMatrix[2] * yin + fMatrix[4]

yout = fMatrix[1] * xin + fMatrix[3] * yin + fMatrix[5]

If fMatrix = NULL an identity matrix (no transformation) will be set. If pectx is equal to NULL mmlGdcPeSetMatrix is terminated without any operation.

Parameters

in,out	pectx	Pixel Engine context (!=NULL).
in	target	Setting target. It is a single or OR combined value of:MML_GDC_PE_SRC MML_GDC_PE_DST MML_GDC_PE_MASK
in	format	Defines the matrix format (see above).
in	fMatrix	Transformation matrix (see above).

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.18 MML_GDC_PE_API MM_ERROR mmlGdcPeSurfAttribute (MML_GDC_PE_CONTEXT pectx, MM_U32 target, MML_GDC_PE_SURF_ATTR pname, MM_U32 param)

Set an attribute for the surface that is bound to the specified target. If pectx is equal to NULL, mmlGdcPeSurfAttribute is terminated without any operation.

Note

- *The MASK surface does not support color multiplication. The function reports an error if a related parameter is set.*

Parameters

in,out	pectx	Pixel Engine context (!=NULL).
in	target	Setting target - MML_GDC_PE_SRC (all attributes). - MML_GDC_PE_DST (all attributes). - MML_GDC_PE_MASK (all attributes). - or MML_GDC_PE_STORE (attribute MML_GDC_PE_SURF_ATTR_USE_CLIPPING only).
in	pname	State name for setting. Can be one of MML_GDC_PE_SURF_ATTR.
in	param	Parameter for target. See MML_GDC_PE_SURF_ATTR description.

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.19 MML_GDC_PE_API MM_ERROR mmlGdcPeSurfColor (MML_GDC_PE_CONTEXT pectx, MM_U32 target, MM_U08 red, MM_U08 green, MM_U08 blue, MM_U08 alpha)

Set the constant color for the specified target. Indicated color is used for following usage. (See also mmlGdcPeSurfAttribute.)

- Constant color in color multiplication.
- Constant color in tiling.
- Constant color used for the generation of a color component in format conversion (e.g., format conversion from RGB565 to RGBA8888 if 0x1234_5678 is used as constant color 0xFFFF (RGB565) -> 0xFFFF_FF78 (RGBA8888)).

If pectx is equal to NULL, mmlGdcPeSurfColor is terminated without any operation.

Parameters

in,out	pectx	Pixel Engine context (!=NULL).
in	target	Setting target MML_GDC_PE_SRC MML_GDC_PE_DST MML_GDC_PE_MASK
in	red	Red component of color (0 - 255, default 255).
in	green	Green component of color (0 - 255, default 255).
in	blue	Blue component of color (0 - 255, default 255).
in	alpha	Alpha component of color (0 - 255, default 255).

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.20 MML_GDC_PE_API MM_ERROR mmlGdcPeSync (MML_GDC_SYNC sync)

Inserts a sync object into the 2D command stream (similar to the OpenGL glFencSync() call).

Parameters

in,out	sync	Sync object reset by mmlGdcSyncReset(). After successful completion of mmlGdcPeSync(), it holds the parameter of the inserted sync.

Return values

MML_OK	On success. Otherwise the related error code.

11.6.5.21 MM_ERROR mmlGdcPeWaitForDispFrameEnd (MML_GDC_DISPLAY display, MM_U32 line)

Delay blit execution until a defined line is passed by the display controller.

mmlGdcPeWaitForDispFrameEnd adds an instruction to the blit and draw command list to wait until the display controller enters a defined line. It can be used to start rendering in the blanking phase or at a defined time point in a single render buffer solution. This function can be called multiple times within a frame to coordinate rendering of different regions.

Parameters

in	display	An MML_GDC_DISPLAY returned from a previous call to mmlGdcDispOpenDisplay().
in	line	The line parameter defines the display line when rendering starts. 0 stands for the first line. The maximal valid line is the vertical resolution i.e. rendering will be continued in the blanking phase.

11.6.5.22 MML_GDC_PE_API MM_ERROR mmlGdcPeWaitSync (MML_GDC_SYNC sync)

Inserts a sync wait into the 2D command stream (similar to the OpenGL glWaitSync() call). PixEngine operations performed after this call are only executed after sync gets signaled.

Parameters

in	sync	Sync to wait for.

Return values

MML_OK	On success. Otherwise the related error code.

11.7 Synchronization API

Synchronization API - Synchronization of framebuffer operations.

Data Structures

- struct MML_GDC_SYNC_CONTAINER

Typedefs

- typedef MML_GDC_SYNC_CONTAINER * MML_GDC_SYNC

Functions

- MM_ERROR mmlGdcSyncReset (MML_GDC_SYNC sync)
- MM_ERROR mmlGdcSyncWait (MML_GDC_SYNC sync, MM_S32 timeout)
- MM_ERROR mmlGdcSyncIncr (MML_GDC_SYNC sync, MM_S32 incr)

11.7.1 Detailed Description

Synchronization API - Synchronization of framebuffer operations.

#include "mml_gdc_sync.h"

The Synchronization API provides mechanisms to synchronize framebuffer operations. These are

- 2D graphics operations (e.g., blt finished).
- Display operations (e.g., framebuffer displayed, VSync happened).

Synchronization is achieved through sync objects - a representation of events whose completion status can be tested or waited upon. Waiting can be done by

- The CPU (see mmlGdcSyncWait()).
- As part of a graphics operation (more details below).

The function to initialize a sync object, (i.e., setting the sync condition, and the function to perform a wait as part of a graphics operation are part of the corresponding module's API):

- 2D operations: See Pixel Engine API.
- Display: See Display API.

11.7.2 Typedef Documentation

11.7.2.1 typedef MML_GDC_SYNC_CONTAINER* MML_GDC_SYNC

The sync object definition.

11.7.3 Function Documentation

11.7.3.1 MM_ERROR mmlGdcSyncIncr (MML_GDC_SYNC sync, MM_S32 incr)

Increments the sync count for sync object sync. This way a sync object can be used to wait for last sync condition + incr. This must only be used for sync sources that increment the sync counter in a known fashion (e.g., display controller VSync)!

Parameters

in	sync	Sync object for which to increment the sync counter.
in	incr	Sync counter increment. Parameter must be -32768 <= incr <= 32767.

Return values

MML_OK	Success.
MML_ERR_GDC_SYNC_INVALID	Sync object not valid.
MML_ERR_GDC_SYNC_INVALID_PARAMETER	Invalid parameter.

11.7.3.2 MM_ERROR mmlGdcSyncReset (MML_GDC_SYNC sync)

Reset all parameters of the sync object.

Parameters

in,out	sync	The sync object.

Return values

MML_OK	On success. Otherwise the related error code.

11.7.3.3 MM_ERROR mmlGdcSyncWait (MML_GDC_SYNC sync, MM_S32 timeout)

Waits for a sync object to be signaled.

Parameters

in	sync	Sync object to wait for getting signaled.
in	timeout	This parameter MUST be 0 for S6E2D.

Return values

MML_OK	Success.
MML_ERR_GDC_SYNC_INVALID_PARAMETER	Invalid parameter.
MML_ERR_GDC_SYNC_INVALID	Sync object not valid.
MML_ERR_GDC_SYNC_TIMEOUT	Sync object is not signaled.

11.8 2D Core Interrupt Controller API

2D Core Interrupt Controller handler functions

Macros

- #define MM_GDC_IRIS_INT_STORE9_FRAMECOMPLETE_IRQ_CP 1U
- #define MM_GDC_IRIS_INT_EXTDST0_FRAMECOMPLETE_IRQ_CP 4U
- #define MM_GDC_IRIS_INT_DISENGCFG_FRAMECOMPLETE0_IRQ_CP 10U
- #define MM_GDC_IRIS_INT_CMDSEQ_ERROR_IRQ_CP 20U
- #define MM_GDC_IRIS_INT_FRAMEGEN0_SECSYNC_ON_IRQ_CP 27U
- #define MM_GDC_IRIS_INT_FRAMEGEN0_SECSYNC_OFF_IRQ_CP 28U

Interrupt signal irqs

These can be used in mmdGdcInterruptRegisterHandler

- #define MM_GDC_IRIS_STORE9_FRAMECOMPLETE_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_STORE9_FRAMECOMPLETE_IRQ_CP)
- #define MM_GDC_IRIS_EXTDST0_FRAMECOMPLETE_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_EXTDST0_FRAMECOMPLETE_IRQ_CP)
- #define MM_GDC_IRIS_DISENGCFG_FRAMECOMPLETE0_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_DISENGCFG_FRAMECOMPLETE0_IRQ_CP)
- #define MM_GDC_IRIS_CMDSEQ_ERROR_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_CMDSEQ_ERROR_IRQ_CP)
- #define MM_GDC_IRIS_FRAMEGEN0_SECSYNC_ON_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_FRAMEGEN0_SECSYNC_ON_IRQ_CP)
- #define MM_GDC_IRIS_FRAMEGEN0_SECSYNC_OFF_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_FRAMEGEN0_SECSYNC_OFF_IRQ_CP)

Interrupt Operations Functions

- void mmdGdcInterruptHandler (void)

  Interrupt Handler Function.
- MM_ERROR mmdGdcInterruptRegisterHandler (MM_U64 irq, void(*pHandler)(MM_U64 intrrpt))

  Set an application defined interrupt handler function.

11.8.1.1 Detailed Description

2D Core Interrupt Controller handler functions

#include "mmd_gdc_interrupthandler.h"

The interrupt controller API provides all required functions to handle 2D core interrupts.

Note:

- *The 2D core interrupts are required for the 2D Core Graphics Driver. Therefore it is required that all 2D core IRQ lines connected to the ARM core are enabled and linked to the mmdGdcInterruptHandler function provided by this interface. The 2D Core Driver will take care that the interrupt sources are reset.*

Optionally it is possible for an application to register a callback function for dedicated 2D core interrupts using

mmdGdcInterruptRegisterHandler. In this case the driver will call the function after clearing the interrupt status.

11.8.2 Macro Definition Documentation

11.8.2.1 #define MM_GDC_IRIS_CMDSEQ_ERROR_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_CMDSEQ_ERROR_IRQ_CP)

CMDSEQ_ERROR: Error condition (Command Sequencer).

11.8.2.2 #define MM_GDC_IRIS_DISENGCFG_FRAMECOMPLETE0_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_DISENGCFG_FRAMECOMPLETE0_IRQ_CP)

DISENGCFG_FRAMECOMPLETE0: Frame complete (Display Controller, Display Stream 0).

11.8.2.3 #define MM_GDC_IRIS_FRAMEGEN0_SECSYNC_OFF_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_FRAMEGEN0_SECSYNC_OFF_IRQ_CP)

FRAMEGEN0_SECSYNC_OFF: Synchronization status deactivated (Display Controller, Content stream 0).

11.8.2.4 #define MM_GDC_IRIS_FRAMEGEN0_SECSYNC_ON_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_FRAMEGEN0_SECSYNC_ON_IRQ_CP)

FRAMEGEN0_SECSYNC_ON: Synchronization status activated (Display Controller, Content stream 0).

11.8.3 Function Documentation

11.8.3.1 void mmdGdcInterruptHandler (void)

Interrupt Handler Function.

Interrupt service routine for 2D Core interrupts. This function has to be called by ARM if any 2D Core interrupt occurs. The function takes care that the interrupt status in the 2D Core is reset. The related interrupt status in ARM must be reset by the calling function.

11.8.3.2 MM_ERROR mmdGdcInterruptRegisterHandler (MM_U64 irq, void(*)(MM_U64 intrrpt) pHandler)

Set an application defined interrupt handler function.

This function allows an application to define a callback function for dedicated interrupts at runtime. The function ensures that the related interrupts are enabled in the 2D Core HW block.

Note

- *The callback function must not call any 2D Core driver APIs as direct action because it is part of the ARM interrupt sequence. The callback function will be called after the driver has handled the interrupt internally.*

Parameters

in	irq	"or"ed Bitmask with all interrupts calling pHandler.
in	pHandler	Callback function that will be called if one or more requested interrupts occur. The MM_U64 parameter indicates the related interrupts. If pHandler is zero the callback function will no longer be called.

Return values

MMD_OK	on success. Otherwise the related error code.

11.9 Error Reporting API

Error Reporting API - Error Reporting for selected modules and level.

Typedefs

- typedef void MM_PRINTFUNCTION (const char ∗string)

Enumerations

- enum MM_ERP_MESSAGE_LEVEL {
  MM_ERP_LEVEL_NOTHING = 0U,
  MM_ERP_LEVEL_ERROR,
  MM_ERP_LEVEL_WARNING, MM_ERP_LEVEL_INFO
  }
- enum MM_ERP_MESSAGE_CHANNEL_PROP {
  MM_ERP_CH_OFF = 0U,
  MM_ERP_CH_ON
  }
- enum MM_ERP_MESSAGE_DEST {
  MM_ERP_CH_STDOUT = 0U,
  MM_ERP_CH_BUFFER
  }

Functions

- MM_ERROR mmlGdcErpSetMessageLevel (MM_U32 moduleId, MM_ERP_MESSAGE_LEVEL level)
- MM_ERROR mmlGdcErpSetMessageChannel (MM_ERP_MESSAGE_DEST dest, MM_ERP_MESSAGE_CHANNEL_PROP prop)
- MM_ERROR mmlGdcErpSetBuffer (MM_ADDR bufferAddr, MM_U32 bufferSize)
- MM_ERROR mmlGdcErpSetPrintf (MM_PRINTFUNCTION ∗user_print_function)

Module Id's

(The error reporting level can be set per module id)

Note

kernel modules are covered by the corresponding user module

- *#define MM_ERP_MODULE_ID_GDC_ALL_USER MM_MODULEID(0x2100FFFFU)*
- *#define MM_ERP_MODULE_ID_GDC_SURFMAN_USER MM_MODULEID(0x21000000U)*
- *#define MM_ERP_MODULE_ID_GDC_DISP_USER MM_MODULEID(0x21001000U)*
- *#define MM_ERP_MODULE_ID_GDC_IRIS_USER MM_MODULEID(0x21003000U)*
- *#define MM_ERP_MODULE_ID_GDC_SYNC_USER MM_MODULEID(0x21005000U)*

- *#define MM_ERP_MODULE_ID_GDC_CARD_USER MM_MODULEID(0x21006000U)*
- *#define MM_ERP_MODULE_ID_GDC_CONFIG_USER MM_MODULEID(0x21007000U)*
- *#define MM_ERP_MODULE_ID_GDC_SYSINIT_USER MM_MODULEID(0x21008000U)*
- *#define MM_ERP_MODULE_ID_GDC_CMDSEQ_USER MM_MODULEID(0x21009000U)*
- *#define MM_ERP_MODULE_ID_GDC_PIXENG_USER MM_MODULEID(0x2100B000U)*
- *#define MM_ERP_MODULE_ID_GDC_ERP_USER MM_MODULEID(0x2100D000U)*
- *#define MM_ERP_MODULE_ID_GDC_SERVICE_USER MM_MODULEID(0x2100E000U)*

11.9.1 Detailed Description

Error Reporting API - Error Reporting for selected modules and level. The module Ids of this driver.

#include "mml_gdc_erp.h"

The Error-Reporting API provides functions to report errors, warnings and infos. The modules that are covered can be specified.

The user has the options to select the level of messages for selectable modules (mmlGdcErpSetMessageLevel), to select the channel of messages (mmlGdcErpSetMessageChannel).

When using channel MM_ERP_CH_STDOUT the configuration of the print function by mmlGdcErpSetPrintf is necessary.

When using channel MM_ERP_CH_BUFFER the configuration of the buffer by mmlGdcErpSetBuffer is necessary.

Note:

- *Error reporting is only available in Debug and Release configuration! In Production configuration this functionality is switched OFF.*

#include "mm_gdc_module_id.h"

The module ids are used to en-/disable message logging for certain modules of the driver. Wildcards can be used to en-/disable messages for all modules of the driver.

For details see mmlGdcErpSetMessageLevel

11.9.2 Macro Definition Documentation

11.9.2.1 #define MM_ERP_MODULE_ID_GDC_ALL_USER MM_MODULEID(0x2100FFFFU)

Wildcard for all modules of basic graphics driver

11.9.2.2 #define MM_ERP_MODULE_ID_GDC_CARD_USER MM_MODULEID(0x21006000U)

Card (HW access)

11.9.2.3 #define MM_ERP_MODULE_ID_GDC_CMDSEQ_USER MM_MODULEID(0x21009000U)

Command Sequencer

11.9.2.4 #define MM_ERP_MODULE_ID_GDC_CONFIG_USER MM_MODULEID(0x21007000U)

Configuration

11.9.2.5 #define MM_ERP_MODULE_ID_GDC_DISP_USER MM_MODULEID(0x21001000U)

Display

11.9.2.6 #define MM_ERP_MODULE_ID_GDC_ERP_USER MM_MODULEID(0x2100D000U)

Error Reporting

11.9.2.7 #define MM_ERP_MODULE_ID_GDC_IRIS_USER MM_MODULEID(0x21003000U)

Internal components

11.9.2.8 #define MM_ERP_MODULE_ID_GDC_PIXENG_USER MM_MODULEID(0x2100B000U)

Pixel Engine

11.9.2.9 #define MM_ERP_MODULE_ID_GDC_SERVICE_USER MM_MODULEID(0x2100E000U)

Resource Manager

11.9.2.10 #define MM_ERP_MODULE_ID_GDC_SURFMAN_USER MM_MODULEID(0x21000000U)

Surface Manager

11.9.2.11 #define MM_ERP_MODULE_ID_GDC_SYNC_USER MM_MODULEID(0x21005000U)

Synchronization

11.9.2.12 #define MM_ERP_MODULE_ID_GDC_SYSINIT_USER MM_MODULEID(0x21008000U)

Initialization

11.9.3 Typedef Documentation

11.9.3.1 typedef void MM_PRINTFUNCTION(const char *string)

Function type definition for the print function that shall be used.

11.9.4 Enumeration Type Documentation

11.9.4.1 enum MM_ERP_MESSAGE_CHANNEL_PROP

Enumeration of message channel properties

Enumerator

MM_ERP_CH_OFF message channel off

MM_ERP_CH_ON message channel on

11.9.4.2 enum MM_ERP_MESSAGE_DEST

Enumeration of message destination

Enumerator

MM_ERP_CH_STDOUT report to stdout

MM_ERP_CH_BUFFER report to buffer

11.9.4.3 enum MM_ERP_MESSAGE_LEVEL

Enumeration of message levels

Enumerator

MM_ERP_LEVEL_NOTHING report no messages

MM_ERP_LEVEL_ERROR report error messages

MM_ERP_LEVEL_WARNING report error+warning messages

MM_ERP_LEVEL_INFO report error+warning+info messages

11.9.5 Function Documentation

11.9.5.1 MM_ERROR mmlGdcErpSetBuffer (MM_ADDR bufferAddr, MM_U32 bufferSize)

Set the parameter for a buffer, that is used as a channel for error messages.

Parameters

in	bufferAddr	Address of the provided buffer
in	bufferSize	Size (in Bytes) of the provided buffer.

Return values

MML_OK	Normal termination.
MML_ERR_ERP_INVALID_PARAMETER	An invalid value is set in an argument.

11.9.5.2 MM_ERROR mmlGdcErpSetMessageChannel (MM_ERP_MESSAGE_DEST dest, MM_ERP_MESSAGE_CHANNEL_PROP prop)

Set channel for error messages. By default only MM_ERP_CH_STDOUT is ON.

Note

- *MM_ERP_CH_STDOUT and MM_ERP_CH_BUFFER can be en-/disabled. independently.*

Parameters

in	dest	Message channel selection: - MM_ERP_CH_STDOUT Messages are routed to stdout. - MM_ERP_CH_BUFFER Messages are routed to a buffer.
in	prop	Setting of specified message channel: - MM_ERP_CH_OFF Set message channel OFF. - MM_ERP_CH_ON Set message channel ON.

Return values

MML_OK	Normal termination.
MML_ERR_ERP_INVALID_PARAMETER	An invalid value is set in an argument.

11.9.5.3 MM_ERROR mmlGdcErpSetMessageLevel (MM_U32 moduleId, MM_ERP_MESSAGE_LEVEL level)

Set level of error messages for an individual module. For example,

mmlGdcErpSetMessageLevel(MM_ERP_MODULE_ID_GDC_DISP_USER, MM_ERP_LEVEL_INFO);

will print all messages (info,warning,error) that come from the display module. The module IDs are defined for each driver component (see Module Id's).

Parameters

in	moduleId	Module ID selection. - MM_ERP_MODULE_ID_GDC_ALL_USER - MM_ERP_MODULE_ID_GDC_SURFMAN_USER - MM_ERP_MODULE_ID_GDC_DISP_USER - MM_ERP_MODULE_ID_GDC_IRIS_USER - MM_ERP_MODULE_ID_GDC_SYNC_USER - MM_ERP_MODULE_ID_GDC_CARD_USER - MM_ERP_MODULE_ID_GDC_CONFIG_USER - MM_ERP_MODULE_ID_GDC_SYSINIT_USER - MM_ERP_MODULE_ID_GDC_CMDSEQ_USER - MM_ERP_MODULE_ID_GDC_PIXENG_USER - MM_ERP_MODULE_ID_GDC_ERP_USER - MM_ERP_MODULE_ID_GDC_SERVICE_USER
in	level	Level selection: - MM_ERP_LEVEL_NOTHING No messages. - MM_ERP_LEVEL_ERROR All error messages. - MM_ERP_LEVEL_WARNING All error and warning messages. - MM_ERP_LEVEL_INFO All error, warning and info messages.

Return values

MML_OK	Normal termination.
MML_ERR_ERP_INVALID_PARAMETER	An invalid value is set in an argument.

11.9.5.4 MM_ERROR mmlGdcErpSetPrintf (MM_PRINTFUNCTION * user_print_function)

Set the print function that is used for the STDOUT channel.

Parameters

in	user_print_function	A Function of type MM_PRINTFUNCTION (function returning "void" of parameter "const char *string") that shall be used to "print" on STDOUT.

Note

- *This will be initialized to NULL (i.e., without setting this function, there will be no messages on STDOUT).*

Return values

MML_OK	Normal termination.
MML_ERR_ERP_INVALID_PARAMETER	An invalid value is set in an argument.

11.10 Error Codes

Error Codes of this driver.

Error codes for Config API

- #define MML_ERR_GDC_CONFIG_INVALID_PARAMETER MM_ERRCODE(0x21008001)
- #define MML_ERR_GDC_CONFIG_INTERNAL_ERROR MM_ERRCODE(0x21008002)
- #define MML_ERR_GDC_CONFIG_INVALID_ADDRESS MM_ERRCODE(0x21008003)

Error codes for Display API

- #define MML_ERR_GDC_DISP_DEVICE_NOT_FOUND MM_ERRCODE(0x21001001)
- #define MML_ERR_GDC_DISP_DISPLAY_ALREADY_OPEN MM_ERRCODE(0x21001002)
- #define MML_ERR_GDC_DISP_INVALID_ARG MM_ERRCODE(0x21001003)
- #define MML_ERR_GDC_DISP_UNSUPPORTED_MODE MM_ERRCODE(0x21001004)
- #define MML_ERR_GDC_DISP_DEVICE_INIT_FAILED MM_ERRCODE(0x21001005)
- #define MML_ERR_GDC_DISP_DEVICE_CLOSE_FAILED MM_ERRCODE(0x21001006)
- #define MML_ERR_GDC_DISP_OUT_OF_SYSTEM_MEMORY MM_ERRCODE(0x21001007)
- #define MML_ERR_GDC_DISP_LAYER_ALREADY_USED MM_ERRCODE(0x21001008)
- #define MML_ERR_GDC_DISP_WRONG_PIXEL_FORMAT MM_ERRCODE(0x21001009)
- #define MML_ERR_GDC_DISP_WRONG_STRIDE MM_ERRCODE(0x21001011)
- #define MML_ERR_GDC_DISP_WRONG_WINDOW MM_ERRCODE(0x21001012)
- #define MML_ERR_GDC_DISP_WRONG_INDEX_WINDOW MM_ERRCODE(0x21001013)
- #define MML_ERR_GDC_DISP_FAILED MM_ERRCODE(0x21001014)
- #define MML_ERR_GDC_DISP_WRONG_YC_WINDOW MM_ERRCODE(0x21001015)
- #define MML_ERR_GDC_DISP_WRONG_TCON_PARAMS MM_ERRCODE(0x21001016)
- #define MML_ERR_GDC_DISP_DISPLAY_MODE_MISSMATCH MM_ERRCODE(0x21001017)
- #define MML_ERR_GDC_DISP_INVALID_SCALING MM_ERRCODE(0x21001018)
- #define MML_ERR_GDC_DISP_INVALID_BLENDING MM_ERRCODE(0x21001019)
- #define MML_ERR_GDC_DISP_INVALID_CLUTDATA MM_ERRCODE(0x2100101a)
- #define MML_ERR_GDC_DISP_INVALID_DIMENSION MM_ERRCODE(0x2100101c)
- #define MML_ERR_GDC_DISP_DEV_BUSY MM_ERRCODE(0x21001020)

Error codes for Error Reporting API

- #define MML_ERR_ERP_ALREADY_INITIALIZED MM_ERRCODE(0x2100F000)
- #define MML_ERR_ERP_NOT_INITIALIZED MM_ERRCODE(0x2100F001)
- #define MML_ERR_ERP_INVALID_PARAMETER MM_ERRCODE(0x2100F002)

Error codes for Pixel Engine API

- #define MML_ERR_GDC_PE_OUT_OF_SPACE MM_ERRCODE(0x2100D001)
- #define MML_ERR_GDC_PE_INVALID_CONTEXT MM_ERRCODE(0x2100D002)
- #define MML_ERR_GDC_PE_INVALID_TARGET MM_ERRCODE(0x2100D003)
- #define MML_ERR_GDC_PE_INVALID_SURFACE_OBJECT MM_ERRCODE(0x2100D004)
- #define MML_ERR_GDC_PE_INVALID_ADDRESS MM_ERRCODE(0x2100D005)
- #define MML_ERR_GDC_PE_INVALID_MATRIX MM_ERRCODE(0x2100D006)
- #define MML_ERR_GDC_PE_INVALID_DIMENSION MM_ERRCODE(0x2100D007)
- #define MML_ERR_GDC_PE_INVALID_STRIDE MM_ERRCODE(0x2100D008)
- #define MML_ERR_GDC_PE_INVALID_BITS_PER_PIXEL MM_ERRCODE(0x2100D009)
- #define MML_ERR_GDC_PE_INVALID_COMPRESSION MM_ERRCODE(0x2100D010)
- #define MML_ERR_GDC_PE_INVALID_RLD_REQUEST MM_ERRCODE(0x2100D011)
- #define MML_ERR_GDC_PE_INVALID_ROP_MODE MM_ERRCODE(0x2100D012)
- #define MML_ERR_GDC_PE_INVALID_SURFACE_PARAM MM_ERRCODE(0x2100D013)
- #define MML_ERR_GDC_PE_INVALID_NO_ACTIVE_AREA MM_ERRCODE(0x2100D014)
- #define MML_ERR_GDC_PE_INVALID_ATTRIBUTE MM_ERRCODE(0x2100D015)
- #define MML_ERR_GDC_PE_INVALID_PARAMETER MM_ERRCODE(0x2100D016)
- #define MML_ERR_GDC_PE_INVALID_OPERATION MM_ERRCODE(0x2100D017)
- #define MML_ERR_GDC_PE_INVALID_MASK_PARAM MM_ERRCODE(0x2100D018)
- #define MML_ERR_GDC_PE_INVALID_SCALING MM_ERRCODE(0x2100D019)

- #define MML_ERR_GDC_PE_INVALID_STORE_COMRESSION MM_ERRCODE(0x2100D020)
- #define MML_ERR_GDC_PE_INVALID_STORE_CLUT MM_ERRCODE(0x2100D021)
- #define MML_ERR_GDC_PE_INVALID_FLOAT MM_ERRCODE(0x2100D023)
- #define MML_ERR_GDC_PE_INVALID_CLUT_OPERATION MM_ERRCODE(0x2100D024)
- #define MML_ERR_GDC_PE_INVALID_YUV_PARAM MM_ERRCODE(0x2100D028)
- #define MML_ERR_GDC_PE_INVALID_COMPRESSION_OPERATION MM_ERRCODE(0x2100D029)

Error codes for Surface Manager API

- #define MML_ERR_GDC_SURF_OUT_OF_SPACE MM_ERRCODE(0x21000001)
- #define MML_ERR_GDC_SURF_OUT_OF_VRAM MM_ERRCODE(0x21000002)
- #define MML_ERR_GDC_SURF_INVALID_SURFACE MM_ERRCODE(0x21000003)
- #define MML_ERR_GDC_SURF_INVALID_FORMAT MM_ERRCODE(0x21000004)
- #define MML_ERR_GDC_SURF_INVALID_FOR_BUFFER_OWNED MM_ERRCODE(0x21000005)
- #define MML_ERR_GDC_SURF_INVALID_ATTRIBUTE MM_ERRCODE(0x21000006)
- #define MML_ERR_GDC_SURF_ERROR_ADDRESS_TRANSLATION MM_ERRCODE(0x21000007)
- #define MML_ERR_GDC_SURF_INVALID_PARAMETER MM_ERRCODE(0x21000008)
- #define MML_ERR_GDC_SURF_INVALID_ADDRESS_ALIGNMENT MM_ERRCODE(0x21000009)

Error codes for Synchronization API

- #define MML_ERR_GDC_SYNC_INVALID_PARAMETER MM_ERRCODE(0x21005001)
- #define MML_ERR_GDC_SYNC_OUT_OF_MEMORY MM_ERRCODE(0x21005002)
- #define MML_ERR_GDC_SYNC_TIMEOUT MM_ERRCODE(0x21005003)
- #define MML_ERR_GDC_SYNC_INVALID MM_ERRCODE(0x21005004)

Error codes for Driver Initialization API

- #define MML_ERR_GDC_SYS_DEVICE_INIT_FAILED MM_ERRCODE(0x21009001)
- #define MML_ERR_GDC_SYS_DEVICE_CLOSE_FAILED MM_ERRCODE(0x21009002)
- #define MML_ERR_GDC_SYS_DEVICE_ALREADY_INITIALIZED MM_ERRCODE(0x21009003)
- #define MML_ERR_GDC_SYS_DEVICE_NOT_YET_INITIALIZED MM_ERRCODE(0x21009004)
- #define MML_ERR_GDC_SYS_DEVICE_INVALID_PARAMETER MM_ERRCODE(0x21009005)
- #define MML_ERR_GDC_SYS_DEVICE_WRONG_ID MM_ERRCODE(0x21009006)

Error codes for Writeback API

- #define MML_ERR_GDC_WB_DEVICE_BUSY MM_ERRCODE(0x21004001)
- #define MML_ERR_GDC_WB_INVALID_PARAMETER MM_ERRCODE(0x21004002)

Error codes for Internal function calls

- #define MML_ERR_GDC_CARD_DEV_NOT_ENABLED MM_ERRCODE(0x21007001)
- #define MML_ERR_GDC_CARD_DEV_ENABLED MM_ERRCODE(0x21007002)
- #define MML_ERR_GDC_CARD_DEV_NOTSUPPORTED MM_ERRCODE(0x21007003)
- #define MML_ERR_GDC_CARD_ACCESS_FAILED MM_ERRCODE(0x21007004)

- #define MML_ERR_GDC_CARD_THREAD_LIMIT MM_ERRCODE(0x21007005)
- #define MML_ERR_GDC_CARD_TIMEOUT_EXPIRED MM_ERRCODE(0x21007006)
- #define MML_ERR_GDC_CARD_DEV_BUSY MM_ERRCODE(0x21007007)
- #define MML_ERR_GDC_IRIS_CMD_SEQ_ARG_ERROR MM_ERRCODE(0x2100B001)
- #define MML_ERR_GDC_IRIS_CMD_SEQ_INVALID_ADDRESS MM_ERRCODE(0x2100B002)
- #define MML_ERR_GDC_IRIS_CMD_SEQ_INVALID_BUFFER_SIZE MM_ERRCODE(0x2100B003)
- #define MML_ERR_GDC_IRIS_CMD_SEQ_FIFO_UNINITIALIZED MM_ERRCODE(0x2100B004)
- #define MML_ERR_GDC_IRIS_CMD_SEQ_COMMAND_QUEUE_FULL MM_ERRCODE(0x2100B005)
- #define MMD_ERR_GDC_DISP_ARG_ERROR MM_ERRCODE(0x11001003)
- #define MML_ERR_GDC_INT_OUT_OF_RANGE MM_ERRCODE(0x21010001)
- #define MMD_ERR_GDC_INT_OUT_OF_RANGE MM_ERRCODE(0x11010001)
- #define MML_ERR_GDC_IRIS_MATH_INVALID_FLOAT MM_ERRCODE(0x21003001)
- #define MML_ERR_GDC_IRIS_MATH_INVALID_MATRIX MM_ERRCODE(0x21003002)
- #define MML_ERR_RES_UNKNOWN MM_ERRCODE(0x2100A000)
- #define MML_ERR_RES_EXCEEDED_MAXIMUM_USAGE MM_ERRCODE(0x2100A001)
- #define MML_ERR_RES_USAGE_COUNT_ZERO MM_ERRCODE(0x2100A002)
- #define MML_ERR_RES_MAN_ALREADY_INITIALIZED MM_ERRCODE(0x2100A003)
- #define MML_ERR_RES_MAN_NOT_INITIALIZED MM_ERRCODE(0x2100A004)
- #define MMD_ERR_GDC_SYNC_INVALID_PARAMETER MM_ERRCODE(0x11005001)
- #define MMD_ERR_GDC_SYNC_ACCESS_FAILED MM_ERRCODE(0x11005002)
- #define MMD_ERR_GDC_SYNC_TIMEOUT MM_ERRCODE(0x11005003)
- #define MMD_ERR_GDC_CARD_DEV_BUSY MM_ERRCODE(0x11007001)
- #define MMD_ERR_GDC_CARD_TIMEOUT_EXPIRED MM_ERRCODE(0x11007002)
- #define MMD_ERR_GDC_CARD_ACCESS_FAILED MM_ERRCODE(0x11007003)
- #define MMD_ERR_GDC_CARD_TIME_INTERVAL MM_ERRCODE(0x11007004)
- #define MMD_ERR_GDC_CARD_DEV_NOTSUPPORTED MM_ERRCODE(0x11007005)

11.10.1 Detailed Description

Error Codes of this driver.

#include "mm_gdc_errors.h"

All used Error Codes for all modules are collected here.

11.10.2 Macro Definition Documentation

11.10.2.1 #define MMD_ERR_GDC_CARD_ACCESS_FAILED MM_ERRCODE(0x11007003)

An unexpected internal error occurred

11.10.2.2 #define MMD_ERR_GDC_CARD_DEV_BUSY MM_ERRCODE(0x11007001)

Access to a device is denied (e.g. because a shadow load request is pending)

11.10.2.3 #define MMD_ERR_GDC_CARD_DEV_NOTSUPPORTED MM_ERRCODE(0x11007005)

Operation not supported for device

11.10.2.4 #define MMD_ERR_GDC_CARD_TIME_INTERVAL MM_ERRCODE(0x11007004)

Time interval for measurement to short

11.10.2.5 #define MMD_ERR_GDC_CARD_TIMEOUT_EXPIRED MM_ERRCODE(0x11007002)

A timeout expired while trying to acquire a resource

11.10.2.6 #define MMD_ERR_GDC_DISP_ARG_ERROR MM_ERRCODE(0x11001003)

Wrong arguments

11.10.2.7 #define MMD_ERR_GDC_INT_OUT_OF_RANGE MM_ERRCODE(0x11010001)

Interrupt id is out of range

11.10.2.8 #define MMD_ERR_GDC_SYNC_ACCESS_FAILED MM_ERRCODE(0x11005002)

An unexpected internal error occurred

11.10.2.9 #define MMD_ERR_GDC_SYNC_INVALID_PARAMETER MM_ERRCODE(0x11005001)

An invalid value is specified in an argument

11.10.2.10 #define MMD_ERR_GDC_SYNC_TIMEOUT MM_ERRCODE(0x11005003)

Timeout expired

11.10.2.11 #define MML_ERR_ERP_ALREADY_INITIALIZED MM_ERRCODE(0x2100F000)

The error manager is already initialized

11.10.2.12 #define MML_ERR_ERP_INVALID_PARAMETER MM_ERRCODE(0x2100F002)

An invalid value is set in an argument

11.10.2.13 #define MML_ERR_ERP_NOT_INITIALIZED MM_ERRCODE(0x2100F001)

The error manager is not initialized

11.10.2.14 #define MML_ERR_GDC_CARD_ACCESS_FAILED MM_ERRCODE(0x21007004)

An unexpected internal error occurred

11.10.2.15 #define MML_ERR_GDC_CARD_DEV_BUSY MM_ERRCODE(0x21007007)

Access to a device is denied (e.g. because a shadow load request is pending)

11.10.2.16 #define MML_ERR_GDC_CARD_DEV_ENABLED MM_ERRCODE(0x21007002)

Device is still enabled

11.10.2.17 #define MML_ERR_GDC_CARD_DEV_NOT_ENABLED MM_ERRCODE(0x21007001)

Device is not enabled

11.10.2.18 #define MML_ERR_GDC_CARD_DEV_NOTSUPPORTED MM_ERRCODE(0x21007003)

Operation not supported for device

11.10.2.19 #define MML_ERR_GDC_CARD_THREAD_LIMIT MM_ERRCODE(0x21007005)

Maximum number of supported threads reached

11.10.2.20 #define MML_ERR_GDC_CARD_TIMEOUT_EXPIRED MM_ERRCODE(0x21007006)

A timeout expired while trying to acquire a resource (Work Item etc.)

11.10.2.21 #define MML_ERR_GDC_CONFIG_INTERNAL_ERROR MM_ERRCODE(0x21008002)

Graphics driver internal error

11.10.2.22 #define MML_ERR_GDC_CONFIG_INVALID_ADDRESS MM_ERRCODE(0x21008003)

An invalid address is specified

11.10.2.23 #define MML_ERR_GDC_CONFIG_INVALID_PARAMETER MM_ERRCODE(0x21008001)

The parameter is wrong

11.10.2.24 #define MML_ERR_GDC_DISP_DEV_BUSY MM_ERRCODE(0x21001020)

Previously requested configuration is not completely set up.

11.10.2.25 #define MML_ERR_GDC_DISP_DEVICE_CLOSE_FAILED MM_ERRCODE(0x21001006)

Hardware device(s) failed to close

11.10.2.26 #define MML_ERR_GDC_DISP_DEVICE_INIT_FAILED MM_ERRCODE(0x21001005)

Hardware device(s) failed initialization

11.10.2.27 #define MML_ERR_GDC_DISP_DEVICE_NOT_FOUND MM_ERRCODE(0x21001001)

The display adapter requested was not found

11.10.2.28 #define MML_ERR_GDC_DISP_DISPLAY_ALREADY_OPEN MM_ERRCODE(0x21001002)

The display being opened was already open.

11.10.2.29 #define MML_ERR_GDC_DISP_DISPLAY_MODE_MISSMATCH MM_ERRCODE(0x21001017)

The display is already opened and the current mode does not fit.

11.10.2.30 #define MML_ERR_GDC_DISP_FAILED MM_ERRCODE(0x21001014)

The operation failed for an unknown reason

11.10.2.31 #define MML_ERR_GDC_DISP_INVALID_ARG MM_ERRCODE(0x21001003)

An invalid argument was passed

11.10.2.32 #define MML_ERR_GDC_DISP_INVALID_BLENDING MM_ERRCODE(0x21001019)

The blend mode is not supported.

11.10.2.33 #define MML_ERR_GDC_DISP_INVALID_CLUTDATA MM_ERRCODE(0x2100101a)

The CLUT data is not valid.

11.10.2.34 #define MML_ERR_GDC_DISP_INVALID_DIMENSION MM_ERRCODE(0x2100101c)

The buffer width or height is not valid.

11.10.2.35 #define MML_ERR_GDC_DISP_INVALID_SCALING MM_ERRCODE(0x21001018)

The scale factor is not supported.

11.10.2.36 #define MML_ERR_GDC_DISP_LAYER_ALREADY_USED MM_ERRCODE(0x21001008)

The requested layer is already being used

11.10.2.37 #define MML_ERR_GDC_DISP_OUT_OF_SYSTEM_MEMORY MM_ERRCODE(0x21001007)

The system is out of memory.

11.10.2.38 #define MML_ERR_GDC_DISP_UNSUPPORTED_MODE MM_ERRCODE(0x21001004)

A display mode was requested that is not supported on the hardware

11.10.2.39 #define MML_ERR_GDC_DISP_WRONG_INDEX_WINDOW MM_ERRCODE(0x21001013)

The layer does not support an indexed color format

11.10.2.40 #define MML_ERR_GDC_DISP_WRONG_PIXEL_FORMAT MM_ERRCODE(0x21001009)

The pixel format is not supported by the display controller

11.10.2.41 #define MML_ERR_GDC_DISP_WRONG_STRIDE MM_ERRCODE(0x21001011)

The stride of the pixel buffer must be a multiple of 64

11.10.2.42 #define MML_ERR_GDC_DISP_WRONG_TCON_PARAMS MM_ERRCODE(0x21001016)

Wrong timing controller parameters

11.10.2.43 #define MML_ERR_GDC_DISP_WRONG_WINDOW MM_ERRCODE(0x21001012)

The feature is not supported by the given window

11.10.2.44 #define MML_ERR_GDC_DISP_WRONG_YC_WINDOW MM_ERRCODE(0x21001015)

The layer does not support a YC format

11.10.2.45 #define MML_ERR_GDC_INT_OUT_OF_RANGE MM_ERRCODE(0x21010001)

Interrupt id is out of range

11.10.2.46 #define MML_ERR_GDC_IRIS_CMD_SEQ_ARG_ERROR MM_ERRCODE(0x2100B001)

cmd_seq Wrong arguments for CmdSeq API

11.10.2.47 #define MML_ERR_GDC_IRIS_CMD_SEQ_COMMAND_QUEUE_FULL MM_ERRCODE(0x2100B005)

Command buffer full

11.10.2.48 #define MML_ERR_GDC_IRIS_CMD_SEQ_FIFO_UNINITIALIZED MM_ERRCODE(0x2100B004)

Command buffer has not been initialized

11.10.2.49 #define MML_ERR_GDC_IRIS_CMD_SEQ_INVALID_ADDRESS MM_ERRCODE(0x2100B002)

Buffer address unaligned

11.10.2.50 #define MML_ERR_GDC_IRIS_CMD_SEQ_INVALID_BUFFER_SIZE MM_ERRCODE(0x2100B003)

Buffer size not aligned

11.10.2.51 #define MML_ERR_GDC_IRIS_MATH_INVALID_FLOAT MM_ERRCODE(0x21003001)

Float value is outside of the processable range

11.10.2.52 #define MML_ERR_GDC_IRIS_MATH_INVALID_MATRIX MM_ERRCODE(0x21003002)

Matrix inversion failed

11.10.2.53 #define MML_ERR_GDC_PE_INVALID_ADDRESS MM_ERRCODE(0x2100D005)

Wrong address (For instance not aligned)

11.10.2.54 #define MML_ERR_GDC_PE_INVALID_ATTRIBUTE MM_ERRCODE(0x2100D015)

Invalid attribute (target) was specified for an argument.

11.10.2.55 #define MML_ERR_GDC_PE_INVALID_BITS_PER_PIXEL MM_ERRCODE(0x2100D009)

Invalid value for BitsPerPixel

11.10.2.56 #define MML_ERR_GDC_PE_INVALID_CLUT_OPERATION MM_ERRCODE(0x2100D024)

A lookup table cannot be used in this mode

11.10.2.57 #define MML_ERR_GDC_PE_INVALID_COMPRESSION MM_ERRCODE(0x2100D010)

The compression of a source buffer cannot be applied

11.10.2.58 #define MML_ERR_GDC_PE_INVALID_COMPRESSION_OPERATION MM_ERRCODE(0x2100D029)

The requested operation with a compressed buffer is not supported

11.10.2.59 #define MML_ERR_GDC_PE_INVALID_CONTEXT MM_ERRCODE(0x2100D002)

Context object invalid

11.10.2.60 #define MML_ERR_GDC_PE_INVALID_DIMENSION MM_ERRCODE(0x2100D007)

Surface dimension is out of range

11.10.2.61 #define MML_ERR_GDC_PE_INVALID_FLOAT MM_ERRCODE(0x2100D023)

A float value exceeds the range supported by hardware

11.10.2.62 #define MML_ERR_GDC_PE_INVALID_MASK_PARAM MM_ERRCODE(0x2100D018)

Required parameter is not supported for mask

11.10.2.63 #define MML_ERR_GDC_PE_INVALID_MATRIX MM_ERRCODE(0x2100D006)

A matrix operation cannot be performed

11.10.2.64 #define MML_ERR_GDC_PE_INVALID_NO_ACTIVE_AREA MM_ERRCODE(0x2100D014)

A blit operation was started but no active area source defined

11.10.2.65 #define MML_ERR_GDC_PE_INVALID_OPERATION MM_ERRCODE(0x2100D017)

The requested operation failed

11.10.2.66 #define MML_ERR_GDC_PE_INVALID_PARAMETER MM_ERRCODE(0x2100D016)

Invalid parameter was specified for an argument.

11.10.2.67 #define MML_ERR_GDC_PE_INVALID_RLD_REQUEST MM_ERRCODE(0x2100D011)

Required fetch unit does not support RLD

11.10.2.68 #define MML_ERR_GDC_PE_INVALID_ROP_MODE MM_ERRCODE(0x2100D012)

Not all surfaces are defined for the specified ROP mode

11.10.2.69 #define MML_ERR_GDC_PE_INVALID_SCALING MM_ERRCODE(0x2100D019)

The scale factor exceeds the hardware capabilities

11.10.2.70 #define MML_ERR_GDC_PE_INVALID_STORE_CLUT MM_ERRCODE(0x2100D021)

Store color lookup table not supported

11.10.2.71 #define MML_ERR_GDC_PE_INVALID_STORE_COMRESSION MM_ERRCODE(0x2100D020)

Unsupported store compression type

11.10.2.72 #define MML_ERR_GDC_PE_INVALID_STRIDE MM_ERRCODE(0x2100D008)

Invalid value for Stride

11.10.2.73 #define MML_ERR_GDC_PE_INVALID_SURFACE_OBJECT MM_ERRCODE(0x2100D004)

Surface object invalid

11.10.2.74 #define MML_ERR_GDC_PE_INVALID_SURFACE_PARAM MM_ERRCODE(0x2100D013)

The requested surface features are not supported

11.10.2.75 #define MML_ERR_GDC_PE_INVALID_TARGET MM_ERRCODE(0x2100D003)

Invalid target

11.10.2.76 #define MML_ERR_GDC_PE_INVALID_YUV_PARAM MM_ERRCODE(0x2100D028)

The YUV surface properties is invalid

11.10.2.77 #define MML_ERR_GDC_PE_OUT_OF_SPACE MM_ERRCODE(0x2100D001)

The system runs out of memory to perform this operation

11.10.2.78 #define MML_ERR_GDC_SURF_ERROR_ADDRESS_TRANSLATION MM_ERRCODE(0x21000007)

Address translation failed.

11.10.2.79 #define MML_ERR_GDC_SURF_INVALID_ADDRESS_ALIGNMENT MM_ERRCODE(0x21000009)

The base address alignment is not suitable for this operation.

11.10.2.80 #define MML_ERR_GDC_SURF_INVALID_ATTRIBUTE MM_ERRCODE(0x21000006)

The given attribute is not supported

11.10.2.81 #define MML_ERR_GDC_SURF_INVALID_FOR_BUFFER_OWNED MM_ERRCODE(0x21000005)

The operation is not allowed for buffer owned surface objects

11.10.2.82 #define MML_ERR_GDC_SURF_INVALID_FORMAT MM_ERRCODE(0x21000004)

The given format is not supported

11.10.2.83 #define MML_ERR_GDC_SURF_INVALID_PARAMETER MM_ERRCODE(0x21000008)

The parameter is wrong.

11.10.2.84 #define MML_ERR_GDC_SURF_INVALID_SURFACE MM_ERRCODE(0x21000003)

Surface object invalid

11.10.2.85 #define MML_ERR_GDC_SURF_OUT_OF_SPACE MM_ERRCODE(0x21000001)

The system runs out of memory to perform this operation

11.10.2.86 #define MML_ERR_GDC_SURF_OUT_OF_VRAM MM_ERRCODE(0x21000002)

The video memory runs out of memory to perform this operation

11.10.2.87 #define MML_ERR_GDC_SYNC_INVALID MM_ERRCODE(0x21005004)

Invalid sync object

11.10.2.88 #define MML_ERR_GDC_SYNC_INVALID_PARAMETER MM_ERRCODE(0x21005001)

Invalid parameter

11.10.2.89 #define MML_ERR_GDC_SYNC_OUT_OF_MEMORY MM_ERRCODE(0x21005002)

Out of memory

11.10.2.90 #define MML_ERR_GDC_SYNC_TIMEOUT MM_ERRCODE(0x21005003)

Timeout expired

11.10.2.91 #define MML_ERR_GDC_SYS_DEVICE_ALREADY_INITIALIZED MM_ERRCODE(0x21009003)

Hardware device is already initialized

11.10.2.92 #define MML_ERR_GDC_SYS_DEVICE_CLOSE_FAILED MM_ERRCODE(0x21009002)

Hardware device failed to close

11.10.2.93 #define MML_ERR_GDC_SYS_DEVICE_INIT_FAILED MM_ERRCODE(0x21009001)

Hardware device failed initialization

11.10.2.94 #define MML_ERR_GDC_SYS_DEVICE_INVALID_PARAMETER MM_ERRCODE(0x21009005)

Invalid parameter

11.10.2.95 #define MML_ERR_GDC_SYS_DEVICE_NOT_YET_INITIALIZED MM_ERRCODE(0x21009004)

Hardware device is not yet initialized

11.10.2.96 #define MML_ERR_GDC_SYS_DEVICE_WRONG_ID MM_ERRCODE(0x21009006)

The software driver is not valid for the hardware

11.10.2.97 #define MML_ERR_GDC_WB_DEVICE_BUSY MM_ERRCODE(0x21004001)

Writeback unit busy

11.10.2.98 #define MML_ERR_GDC_WB_INVALID_PARAMETER MM_ERRCODE(0x21004002)

Invalid parameter was specified

11.10.2.99 #define MML_ERR_RES_EXCEEDED_MAXIMUM_USAGE MM_ERRCODE(0x2100A001)

resource cannot be acquired as it has already maximum usage count

11.10.2.100 #define MML_ERR_RES_MAN_ALREADY_INITIALIZED MM_ERRCODE(0x2100A003)

The Resource Manager is already initialized

11.10.2.101 #define MML_ERR_RES_MAN_NOT_INITIALIZED MM_ERRCODE(0x2100A004)

The Resource Manager had not been initialized

11.10.2.102 #define MML_ERR_RES_UNKNOWN MM_ERRCODE(0x2100A000)

unknown resource

11.10.2.103 #define MML_ERR_RES_USAGE_COUNT_ZERO MM_ERRCODE(0x2100A002)

resource cannot be released, as usage count is already zero

11.11 Basic Graphics Type Definitions

Definition of types used in Basic Graphics.

Definition of types used in Basic Graphics.

#include "mml_gdc_types.h"

11.12 Version Numbers

The Version numbers of this driver.

Macros

- #define MM_GDC_MAJOR_VERSION 1U
- #define MM_GDC_MINOR_VERSION 0U

11.12.1 Detailed Description

The Version numbers of this driver.

#include "mm_gdc_version.h"

11.12.2 Macro Definition Documentation

11.12.2.1 #define MM_GDC_MAJOR_VERSION 1U

Major version of the driver.

11.12.2.2 #define MM_GDC_MINOR_VERSION 0U

Minor version of the driver.

11.13 Type Definition

Typedefs

- typedef unsigned char MM_U08
- typedef signed char MM_S08
- typedef unsigned short MM_U16
- typedef signed short MM_S16
- typedef unsigned int MM_U32
- typedef signed int MM_S32
- typedef unsigned long long MM_U64
- typedef signed long long MM_S64
- typedef char MM_CHAR
- typedef float MM_FLOAT
- typedef double MM_DOUBLE
- typedef int MM_BOOL
- typedef unsigned int MM_ADDR
- typedef MM_S32 MM_ERROR
- typedef MM_S32 MM_MODULE

11.13.1 Detailed Description

11.13.2 Typedef Documentation

11.13.2.1 typedef unsigned int MM_ADDR

physical memory address

11.13.2.2 typedef int MM_BOOL

boolean

11.13.2.3 typedef char MM_CHAR

string character

11.13.2.4 typedef double MM_DOUBLE

64-bit IEEE float

11.13.2.5 typedef MM_S32 MM_ERROR

function return code

11.13.2.6 typedef float MM_FLOAT

32-bit IEEE float

11.13.2.7 typedef MM_S32 MM_MODULE

module id

11.13.2.8 typedef signed char MM_S08

signed 8-bit integer

11.13.2.9 typedef signed short MM_S16

signed 16-bit integer

11.13.2.10 typedef signed int MM_S32

signed 32-bit integer

11.13.2.11 typedef signed long long MM_S64

signed 64-bit integer

11.13.2.12 typedef unsigned char MM_U08

unsigned 8-bit integer

11.13.2.13 typedef unsigned short MM_U16

unsigned 16-bit integer

11.13.2.14 typedef unsigned int MM_U32

unsigned 32-bit integer

11.13.2.15 typedef unsigned long long MM_U64

unsigned 64-bit integer

11.14 Macro Definition

Macros

- #define MM_ERRCODE(err) ((MM_ERROR)(err))
- #define MM_MODULEID(moduleId) ((MM_MODULE)(moduleId))
- #define MML_ERR MM_ERRCODE(0x3FFFFFFF)
- #define MMD_ERR MM_ERRCODE(0x7FFFFFFF)
- #define MML_OK MM_ERRCODE(0x0)
- #define MMD_OK MM_ERRCODE(0x0)
- #define MM_FALSE ((MM_BOOL) 0)
- #define MM_TRUE ((MM_BOOL) 1)
- #define NULL ((void ∗)0)
- #define MM_BIT(x) (1U<<(x))
- #define MM_PTR_TO_ADDR(x) (MM_ADDR)(x)
- #define MM_ADDR_TO_PTR(x) (void∗)(x)
- #define MM_ADDR_TO_UINT32(x) (MM_U32)(x)
- #define MM_UINT32_TO_ADDR(x) (MM_ADDR)(x)
- #define MM_PTR_TO_UINT32(x) (MM_U32)(x)
- #define MM_UINT32_TO_PTR(x) (void∗)(x)
- #define MM_ADDR_TO_UINT32PTR(x) (MM_U32∗)(x)
- #define MM_ADDR_TO_SINT32PTR(x) (MM_S32∗)(x)
- #define MM_IO_IRIS_SUBSYSTEM 0xD0A00000U
- #define MM_IO_IRIS_CORE 0xD0A10000U
- #define NULL_FUNCTION ((void) 0)
- #define UNUSED_PARAMETER(x) (void)(x)

11.14.1 Detailed Description

11.14.2 Macro Definition Documentation

11.14.2.1 #define MM_ADDR_TO_PTR(x) (void∗)(x)

Conversion: "MM_ADDR" to "void∗"

11.14.2.2 #define MM_ADDR_TO_SINT32PTR(x) (MM_S32∗)(x)

Conversion: ’MM_ADDR’ to ’MM_S32∗’

11.14.2.3 #define MM_ADDR_TO_UINT32(x) (MM_U32)(x)

Conversion: ’MM_ADDR’ to ’MM_U32’

11.14.2.4 #define MM_ADDR_TO_UINT32PTR(x) (MM_U32∗)(x)

Conversion: ’MM_ADDR’ to ’MM_U32∗’

11.14.2.5 #define MM_BIT(x) (1U<<(x))

Set bit

11.14.2.6 #define MM_ERRCODE(err) ((MM_ERROR)(err))

Macro to define the returned Error Code of the driver function

11.14.2.7 #define MM_FALSE ((MM_BOOL) 0)

Definition of FALSE for bool types

11.14.2.8 #define MM_IO_IRIS_CORE 0xD0A10000U

Graphics Core Base Address

11.14.2.9 #define MM_IO_IRIS_SUBSYSTEM 0xD0A00000U

Graphics Subsystem Base Address

11.14.2.10 #define MM_MODULEID(moduleId)

((MM_MODULE)(moduleId))

Macro to define the IDs of the driver modules

11.14.2.11 #define MM_PTR_TO_ADDR(x) (MM_ADDR)(x)

Conversion: void∗ to MM_ADDR

11.14.2.12 #define MM_PTR_TO_UINT32(x) (MM_U32)(x)

Conversion: 'void∗' to 'MM_U32'

11.14.2.13 #define MM_TRUE ((MM_BOOL) 1)

Definition of TRUE for bool types

11.14.2.14 #define MM_UINT32_TO_ADDR(x) (MM_ADDR)(x)

Conversion: 'MM_U32' to 'MM_ADDR'

11.14.2.15 #define MM_UINT32_TO_PTR(x) (void∗)(x)

Conversion: 'MM_U32' to 'void∗'

11.14.2.16 #define MMD_ERR MM_ERRCODE(0x7FFFFFFF)

Abnormal termination (kernel space)

11.14.2.17 #define MMD_OK MM_ERRCODE(0x0)

Normal termination (kernel space)

11.14.2.18 #define MML_ERR MM_ERRCODE(0x3FFFFFFF)

Abnormal termination (user space)

11.14.2.19 #define MML_OK MM_ERRCODE(0x0)

Normal termination (user space)

11.14.2.20 #define NULL ((void *)0)

Definition of NULL pointer

11.14.2.21 #define NULL_FUNCTION ((void) 0)

Helper macro for deactivated functions

11.14.2.22 #define UNUSED_PARAMETER(x) (void)(x)

Helper macro for unused parameters

11.15 Tutorial Utility Library

Modules

- Utilities for the Memory Management
- Utility functions for matrix calculations
- Utilities for the compatibility with other drivers
- Utilities for the Surface Management
- Utilities for the compression
- Util class collection

11.15.1 Detailed Description

The Utility Library contains many functions to simplify applications or to show the usage of 2D core features.

11.16 Utilities for the Memory Management

Macros

- #define MML_ERR_MMAN_INVALID_PARAMETER MM_ERRCODE(0x18010001)
- #define MML_ERR_MMAN_NO_MEMORY MM_ERRCODE(0x18010002)
- #define MML_ERR_MMAN_NO_VRAM MM_ERRCODE(0x18010003)
- #define MML_ERR_MMAN_INVALID_MEMORY MM_ERRCODE(0x18010004)
- #define MML_ERR_MMAN_ACCESS_FAILED MM_ERRCODE(0x18010005)
- #define MM_VRAM_BASE 0xD0000000U
- #define MM_VRAM_SIZE 0x00080000U
- #define MM_SDRAM_BASE 0xB0080000U

- #define MM_SDRAM_SIZE 0x01000000U

Typedefs

- typedef void * MML_MMAN_HEAP_HANDLE

Functions

- MM_ERROR utMmanReset (void)
- MM_ERROR utMmanCreateHeap (MML_MMAN_HEAP_HANDLE *hdlmem, MM_U32 size, MM_U32 baseAddress)
- MM_ERROR utMmanDestroyHeap (MML_MMAN_HEAP_HANDLE hdlmem)
- MM_ERROR utMmanHeapAlloc (MML_MMAN_HEAP_HANDLE hdlmem, MM_U32 size, MM_U32 alignment, MM_ADDR *addr)
- MM_ERROR utMmanHeapFree (MML_MMAN_HEAP_HANDLE hdlmem, void *addr)
- MM_ERROR utMmanGetSize (MML_MMAN_HEAP_HANDLE hdlmem, MM_U32 *size)
- MM_ERROR utMmanGetFree (MML_MMAN_HEAP_HANDLE hdlmem, MM_U32 *size)
- MM_ERROR utMmanGetLargest (MML_MMAN_HEAP_HANDLE hdlmem, MM_U32 *size)

11.16.1 Detailed Description

This function group is used by many tutorial samples to manage video memory (VRAM) allocation and freeing.

11.16.2 Macro Definition Documentation

11.16.2.1 #define MM_SDRAM_BASE 0xB0080000U

SDRAM Base Address .

11.16.2.2 #define MM_SDRAM_SIZE 0x01000000U

Size of external SDRAM (16 MB on Starter Kit).

11.16.2.3 #define MM_VRAM_BASE 0xD0000000U

VRAM Base Address.

11.16.2.4 #define MM_VRAM_SIZE 0x00080000U

Size of embedded VRAM (512 KB).

11.16.2.5 #define MML_ERR_MMAN_ACCESS_FAILED MM_ERRCODE(0x18010005)

Access failed.

11.16.2.6 #define MML_ERR_MMAN_INVALID_MEMORY MM_ERRCODE(0x18010004)

Address points to an unknown memory block.

11.16.2.7 #define MML_ERR_MMAN_INVALID_PARAMETER MM_ERRCODE(0x18010001)

Wrong argument specified.

11.16.2.8 #define MML_ERR_MMAN_NO_MEMORY MM_ERRCODE(0x18010002)

Out of memory (system).

11.16.2.9 #define MML_ERR_MMAN_NO_VRAM MM_ERRCODE(0x18010003)

Out of memory (VRAM).

11.16.3 Typedef Documentation

11.16.3.1 typedef void* MML_MMAN_HEAP_HANDLE

Type definition for memory heap handle.

11.16.4 Function Documentation

11.16.4.1 MM_ERROR utMmanCreateHeap (MML_MMAN_HEAP_HANDLE * hdlmem, MM_U32 size, MM_U32 baseAddress)

Creates a video memory heap.

Note:

- *Typically, an application would not use this function, but call mmlGdcVideoAlloc() instead, which uses the build in memory heap.*

Parameters

out	hdlmem	Handle to newly created heap.
in	size	Size of heap video memory.
in	baseAddress	Physical start address of heap video memory.

Return values

MML_OK	On success.
MML_ERR_MMAN_NO_MEMORY	If not enough system memory for internal data.

11.16.4.2 MM_ERROR utMmanDestroyHeap (MML_MMAN_HEAP_HANDLE hdlmem)

Destroys a video memory heap.

Note:

- *Typically, an application would not use this function (see utMmanCreateHeap).*

Parameters

in	hdlmem	Handle to heap.

Return values

MML_OK	On success.
MML_ERR_MMAN_INVALID_PARAMETER	If hdlmem is NULL.

11.16.4.3 MM_ERROR utMmanGetFree (MML_MMAN_HEAP_HANDLE hdlmem, MM_U32 * size)

Get the total amount of free memory on the heap.

Note:

- *Typically, an application would not use this function, but call mmlGdcVideoGetFreeTotal()*

Parameters

in	hdlmem	Heap to get information for.
out	size	Pointer to variable to receive the information.

Return values

MML_OK	On success.
MML_ERR_MMAN_INVALID_PARAMETER	If hdlmem is NULL

11.16.4.4 MM_ERROR utMmanGetLargest (MML_MMAN_HEAP_HANDLE hdlmem, MM_U32 * size)

Get the size of the largest free contiguous memory block on the heap.

Note:

- *Typically, an application would not use this function, but call mmlGdcVideoGetLargestBlock()*

Parameters

in	hdlmem	Heap to get information for.
out	size	Pointer to variable to receive the information.

Return values

MML_OK	On success.
MML_ERR_MMAN_INVALID_PARAMETER	If hdlmem is NULL.

11.16.4.5 MM_ERROR utMmanGetSize (MML_MMAN_HEAP_HANDLE hdlmem, MM_U32 * size)

Get the size of the heap.

Note:

- *Typically, an application would not use this function, but call mmlGdcVideoGetSize()*

Parameters

in	hdlmem	Heap to get information for.
out	size	Pointer to variable to receive the information.

Return values

MML_OK	On success.
MML_ERR_MMAN_INVALID_PARAMETER	If hdlmem is NULL.

11.16.4.6 MM_ERROR utMmanHeapAlloc (MML_MMAN_HEAP_HANDLE hdlmem, MM_U32 size, MM_U32 alignment, MM_ADDR * addr)

Allocate a block of memory from the specified heap.

Note:

- *Typically, an application would not use this function (see utMmanCreateHeap).*

Parameters

in	hdlmem	Heap to perform the allocation from.
in	size	Number of bytes to allocate.
in	alignment	Alignment to use for the allocation.
out	addr	Pointer to the newly allocated memory.

Return values

MML_OK	On success.
MML_ERR_MMAN_INVALID_PARAMETER	If a parameter is invalid.
MML_ERR_MMAN_NO_MEMORY	If not enough system memory for internal data.
MML_ERR_MMAN_NO_VRAM	If no contiguous block of size bytes with alignment is available.

11.16.4.7 MM_ERROR utMmanHeapFree

(MML_MMAN_HEAP_HANDLE hdlmem, void * addr)

Free a block of memory previously allocated by utMmanHeapAlloc.

Note:

- *Typically, an application would not use this function (see utMmanCreateHeap).*

Parameters

in	hdlmem	Heap to perform the free from.
in	addr	Pointer to the memory to free.

Return values

MML_OK	On success.
MML_ERR_MMAN_INVALID_PARAMETER	If hdlmem is NULL.
MML_ERR_MMAN_INVALID_MEMORY	If addr does not point to a currently allocated memory block.

11.16.4.8 MM_ERROR utMmanReset (void)

Reset build in memory heap(s).

This function must be called before mmlGdcVideoAlloc(), etc. are called.

Return values

MML_OK	on success
MML_ERR_MMAN_NO_MEMORY	if not enough system memory for internal data.

11.17 Utility functions for matrix calculations

Macros

- #define MML_GDC_2D_MATRIX_API extern

Typedefs

- typedef MM_FLOAT Mat3x2 [6]
- typedef MM_FLOAT Mat3x3 [9]
- typedef MM_FLOAT Mat4x4 [16]
- typedef MM_FLOAT Mat4x3 [12]
- typedef MM_FLOAT Mat5x4 [20]

Matrix functions for geometric operations

- MML_GDC_2D_MATRIX_API void utMat3x2Copy (Mat3x2 dst, const Mat3x2 src)
- MML_GDC_2D_MATRIX_API void utMat3x2Multiply (Mat3x2 dst, const Mat3x2 src1, const Mat3x2 src2)
- MML_GDC_2D_MATRIX_API void utMat3x2LoadIdentity (Mat3x2 m)
- MML_GDC_2D_MATRIX_API void utMat3x2Translate (Mat3x2 m, MM_FLOAT x, MM_FLOAT y)
- MML_GDC_2D_MATRIX_API void utMat3x2TranslatePre (Mat3x2 m, MM_FLOAT x, MM_FLOAT y)

- MML_GDC_2D_MATRIX_API void utMat3x2Scale (Mat3x2 m, MM_FLOAT x, MM_FLOAT y)
- MML_GDC_2D_MATRIX_API void utMat3x2ScalePre (Mat3x2 m, MM_FLOAT x, MM_FLOAT y)
- MML_GDC_2D_MATRIX_API void utMat3x2Rot (Mat3x2 m, MM_FLOAT f)
- MML_GDC_2D_MATRIX_API void utMat3x2RotPre (Mat3x2 m, MM_FLOAT f)
- MML_GDC_2D_MATRIX_API MM_U32 utMat3x2Invert (Mat3x2 m)
- MML_GDC_2D_MATRIX_API void utMat3x2GetXY (const Mat3x2 m, const MM_FLOAT x, const MM_FLOAT y, MM_FLOAT ∗xout, MM_FLOAT ∗yout)
- MML_GDC_2D_MATRIX_API void utMat3x3LoadIdentity (Mat3x3 m)
- MML_GDC_2D_MATRIX_API void utMat3x3Copy (Mat3x3 dst, const Mat3x3 src)
- MML_GDC_2D_MATRIX_API void utMat3x3Multiply (Mat3x3 dst, const Mat3x3 src1, const Mat3x3 src2)
- MML_GDC_2D_MATRIX_API void utMat3x3Translate (Mat3x3 m, MM_FLOAT x, MM_FLOAT y)
- MML_GDC_2D_MATRIX_API void utMat3x3TranslatePre (Mat3x3 m, MM_FLOAT x, MM_FLOAT y)
- MML_GDC_2D_MATRIX_API void utMat3x3Scale (Mat3x3 m, MM_FLOAT x, MM_FLOAT y)
- MML_GDC_2D_MATRIX_API void utMat3x3RotX (Mat3x3 m, MM_FLOAT f)
- MML_GDC_2D_MATRIX_API void utMat3x3RotZ (Mat3x3 m, MM_FLOAT f)
- MML_GDC_2D_MATRIX_API void utMat4x4Copy (Mat4x4 dst, const Mat4x4 src)
- MML_GDC_2D_MATRIX_API void utMat4x4Multiply (Mat4x4 dst, const Mat4x4 src1, const Mat4x4 src2)
- MML_GDC_2D_MATRIX_API void utMat4x4LoadIdentity (Mat4x4 m)
- MML_GDC_2D_MATRIX_API void utMat4x4Translate (Mat4x4 m, MM_FLOAT x, MM_FLOAT y, MM_FLOAT z)
- MML_GDC_2D_MATRIX_API void utMat4x4Scale (Mat4x4 m, MM_FLOAT x, MM_FLOAT y, MM_FLOAT z)
- MML_GDC_2D_MATRIX_API void utMat4x4RotX (Mat4x4 m, MM_FLOAT f)
- MML_GDC_2D_MATRIX_API void utMat4x4RotY (Mat4x4 m, MM_FLOAT f)
- MML_GDC_2D_MATRIX_API void utMat4x4RotZ (Mat4x4 m, MM_FLOAT f)
- MML_GDC_2D_MATRIX_API void utMat4x4Perspective (Mat4x4 m, MM_FLOAT fovy, MM_FLOAT aspect, MM_FLOAT zNear, MM_FLOAT zFar)
- MML_GDC_2D_MATRIX_API void utMat4x4GetXYZ (Mat4x4 m, float x, float y, float z, float ∗xout, float ∗yout, float ∗zout)

Matrix functions for the conversion of matrices

- MML_GDC_2D_MATRIX_API void utMat3x2ToMat4x4 (Mat3x2 src, Mat4x4 dst)
- MML_GDC_2D_MATRIX_API void utMat3x3ToMat4x4 (Mat3x3 src, Mat4x4 dst)
- MML_GDC_2D_MATRIX_API void utMat4x4ToMat3x3 (Mat4x4 src, Mat3x3 dst)
- MML_GDC_2D_MATRIX_API void utMat4x4ToMat3x2 (Mat4x4 src, Mat3x2 dst)

Matrix functions for color operations

- MML_GDC_2D_MATRIX_API void utMat4x3Copy (Mat4x3 dst, const Mat4x3 src)
- MML_GDC_2D_MATRIX_API void utMat4x3Multiply (Mat4x3 dst, const Mat4x3 src1, const Mat4x3 src2)
- MML_GDC_2D_MATRIX_API void utMat4x3LoadIdentity (Mat4x3 m)
- MML_GDC_2D_MATRIX_API void utMat5x4LoadIdentity (Mat5x4 m)
- MML_GDC_2D_MATRIX_API void utMat4x3CalcColMatrix (Mat4x3 dst, MM_FLOAT fContrast, MM_FLOAT fBrightness, MM_FLOAT fSaturation, MM_FLOAT fHue)

11.17.1 Detailed Description

The functions in this block are used by some tutorial examples for matrix operations. Different matrix formats and related functions are defined to support different use cases:

- Mat3x2: This matrix format is sufficient for a fine operations like translation, rotation, scaling and sharing.
- Mat3x3: A 3x3 matrix is required for perspective operations.
- Mat4x4: A 4x4 matrix is required for perspective operations including z (depth) calculation. The 2D core HW does not calculate z-coordinates and the driver API does not support this type of matrix. However, in the computer 3D world (e.g. OpenGL) 4x4 matrices are often used and for compatibility reasons in some tutorial examples the 4x4 matrix type is used. To use such a matrix type with the 2D core HW it is required to convert this matrix to a 3x3 matrix type and in some cases to make the z-order calculation in software.
- Mat4x3: This matrix type is useful for color operations, modifying the R, G, B or Y, U, V color channels.
- Mat5x4: This matrix type is useful for color operations, modifying the R, G, B, A or Y, U, V, A color channels.

The following code shows a matrix calculation using typical 3D operations with a 4x4 matrix. The result will be converted into a 3x3 matrix and assigned to a blit context.

```
Mat4x4 m44;
Mat3x3 m33;
utMat4x4LoadIdentity(m44); utMat4x4Translate(m44, w / 2.0f, h / 2.0f, 0);
utMat4x4Scale(m44, w / 4.0f, h / 4.0f, 1); utMat4x4Perspective(m44, 60.0f, (float)w / h, (float)0.1, 100.0);
utMat4x4Translate(m44, 0, 0, -2);
utMat4x4RotX(m44,  40);//fAngle);
utMat4x4RotZ(m44,  30);//fAngle2);
utMat4x4Scale(m44, (float)2 / iw, (float)2 / ih, 1);
utMat4x4Translate(m44, -iw / 2.0f, -ih/2.0f, 0);

//utMat4x4Trace("M4x4",  m44);
utMat4x4ToMat3x3(m44, m33);

//utMat3x3Trace("M3x3",  m33);
mmlGdcPeSetMatrix(ctx,  MML_GDC_PE_SRC, MML_GDC_PE_GEO_MATRIX_FORMAT_3X3, m33);
```

11.17.2 Macro Definition Documentation

11.17.2.1 #define MML_GDC_2D_MATRIX_API extern

MML_GDC_2D_MATRIX_API can be used to define function types like dll export.

11.17.3 Typedef Documentation

11.17.3.1 typedef MM_FLOAT Mat3x2[6]

Matrix with 3 columns and 2 rows for a fine geometry operations. If m is a Mat3x2 matrix type and x,y in an input vector the resulting vector is:

$$\begin{pmatrix} m[0] & m[2] & m[4] \\ m[1] & m[3] & m[5] \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} (m[0] \times x) + (m[2] \times y) + m[4] \\ (m[1] \times x) + (m[3] \times y) + m[5] \end{pmatrix}$$

11.17.3.2 typedef MM_FLOAT Mat3x3[9]

Matrix with 3 columns and 3 rows for perspective geometry operations. If m is a Mat3x3 matrix type and x,y in an input vector the resulting vector is:

$$\begin{pmatrix} m[0] & m[3] & m[6] \\ m[1] & m[4] & m[7] \\ m[2] & m[5] & m[8] \end{pmatrix} \times \begin{pmatrix} y \\ x \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{m[0] \times x + m[3] \times y + m[6]}{m[2] \times x + m[5] \times y + m[8]} \\ \frac{m[1] \times x + m[4] \times y + m[7]}{m[2] \times x + m[5] \times y + m[8]} \\ 1 \end{pmatrix}$$

11.17.3.3 typedef MM_FLOAT Mat4x3[12]

Matrix with 4 columns and 3 rows for color operations with the R, G, B or Y, U, V channels. If m is a Mat4x3 matrix type and R, G, B in an input vector the resulting vector is:

$$\begin{pmatrix} m[0] & m[3] & m[6] & m[9] \\ m[1] & m[4] & m[7] & m[12] \\ m[2] & m[5] & m[8] & m[11] \end{pmatrix} \times \begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} m[0] \times R + m[3] \times G + m[6] \times B + m[9] \\ m[1] \times R + m[4] \times G + m[7] \times B + m[10] \\ m[2] \times R + m[5] \times G + m[8] \times B + m[11] \end{pmatrix}$$

11.17.3.4 typedef MM_FLOAT Mat4x4[16]

Matrix with 4 columns and 4 rows for perspective geometry operations including z calculation. The Mat4x4 matrix is defined in the following order

$$\begin{pmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{pmatrix}$$

11.17.3.5 typedef MM_FLOAT Mat5x4[20]

Matrix with 5 columns and 4 rows for color operations with the R, G, B, A or Y, U, V, A channels. If m is a Mat5x4 matrix type and R, G, B, A in an input vector the resulting vector is:

$$\begin{pmatrix} m[0] & m[4] & m[8] & m[12] & m[16] \\ m[1] & m[5] & m[9] & m[13] & m[17] \\ m[2] & m[6] & m[10] & m[14] & m[18] \\ m[3] & m[7] & m[11] & m[15] & m[19] \end{pmatrix} \times \begin{pmatrix} R \\ G \\ B \\ A \end{pmatrix} = \begin{pmatrix} m[0] \times R + m[4] \times G + m[8] \times B + m[12] \times A + m[16] \\ m[1] \times R + m[5] \times G + m[9] \times B + m[13] \times A + m[17] \\ m[2] \times R + m[6] \times G + m[10] \times B + m[14] \times A + m[18] \\ m[3] \times R + m[7] \times G + m[11] \times B + m[15] \times A + m[19] \end{pmatrix}$$

11.17.4 Function Documentation

11.17.4.1 MML_GDC_2D_MATRIX_API void utMat3x2Copy (Mat3x2 dst, const Mat3x2 src)

Copy the matrix content to a new one.

Parameters

out	dst	The destination matrix.
in	src	The source matrix.

11.17.4.2 MML_GDC_2D_MATRIX_API void utMat3x2GetXY (const Mat3x2 m, const MM_FLOAT x, const MM_FLOAT y, MM_FLOAT * xout, MM_FLOAT * yout)

Calculate the target position for a given matrix and source position.

Parameters

in	m	The matrix.
in	x	Source x position.
in	y	Source y position.
out	xout	Pointer to the destination x position.
out	yout	Pointer to the destination y position.

11.17.4.3 MML_GDC_2D_MATRIX_API MM_U32 utMat3x2Invert (Mat3x2 m)

Calculate the inverted matrix.

Parameters

in,out	m	The matrix to modify.

Return values

MML_OK	On success. Otherwise The related error code

11.17.4.4 MML_GDC_2D_MATRIX_API void utMat3x2LoadIdentity (Mat3x2 m)

Reset the matrix content to a unit matrix.

Parameters

in,out	m	The matrix to modify.

11.17.4.5 MML_GDC_2D_MATRIX_API void utMat3x2Multiply (Mat3x2 dst, const Mat3x2 src1, const Mat3x2 src2)

Multiply 2 matrices. The resulting matrix represents dst = src1 * src2.

Parameters

out	dst	The destination matrix.
in	src1	The first source matrix.
in	src2	The second source matrix.

11.17.4.6 MML_GDC_2D_MATRIX_API void utMat3x2Rot (Mat3x2 m, MM_FLOAT f)

Modify a matrix to realize a rotation. The resulting matrix represents m = m ∗ m_rot.

Parameters

in,out	m	The matrix to modify.
in	f	Rotation angle in degrees.

11.17.4.7 MML_GDC_2D_MATRIX_API void utMat3x2RotPre (Mat3x2 m, MM_FLOAT f)

Modify a matrix by pre-multiplying a rotation matrix. The resulting matrix represents m = m_rot ∗ m.

Parameters

in,out	m	The matrix to modify.
in	f	Rotation angle in degrees.

11.17.4.8 MML_GDC_2D_MATRIX_API void utMat3x2Scale (Mat3x2 m, MM_FLOAT x, MM_FLOAT y)

Modify a matrix to realize a scale operation. The resulting matrix represents m = m ∗ m_scale.

Parameters

in,out	m	The matrix to modify.
in	x	Scale factor in x direction.
in	y	Scale factor in y direction.

11.17.4.9 MML_GDC_2D_MATRIX_API void utMat3x2ScalePre (Mat3x2 m, MM_FLOAT x, MM_FLOAT y)

Modify a matrix by pre-multiplying a scale matrix. The resulting matrix represents m = m_scale ∗ m.

Parameters

in,out	m	The matrix to modify.
in	x	Scale factor in x direction.
in	y	Scale factor in y direction.

11.17.4.10 MML_GDC_2D_MATRIX_API void utMat3x2ToMat4x4 (Mat3x2 src, Mat4x4 dst)

Convert a 3x2-matrix to a 4x4-matrix.

Parameters

out	dst	The destination matrix.
in	src	The source matrix.

11.17.4.11 MML_GDC_2D_MATRIX_API void utMat3x2Translate (Mat3x2 m, MM_FLOAT x, MM_FLOAT y)

Modify a matrix to realize a move operation. The resulting matrix represents m = m ∗ m_trans

Parameters

in,out	m	The matrix to modify.
in	x	Move dimension in x direction.
in	y	Move dimension in y direction.

11.17.4.12 MML_GDC_2D_MATRIX_API void utMat3x2TranslatePre (Mat3x2 m, MM_FLOAT x, MM_FLOAT y)

Modify a matrix by pre-multiplying a move matrix. The resulting matrix represents m = m_trans ∗ m.

Parameters

in,out	m	The matrix to modify.
in	x	Move dimension in x direction.
in	y	Move dimension in y direction.

11.17.4.13 MML_GDC_2D_MATRIX_API void utMat3x3Copy (Mat3x3 dst, const Mat3x3 src)

Copy the content of a 3x3-matrix to a new one.

Parameters

out	dst	The destination matrix.
in	src	The source matrix.

11.17.4.14 MML_GDC_2D_MATRIX_API void utMat3x3LoadIdentity (Mat3x3 m)

Fill a 3x3-matrix with a unit matrix.

Parameters

in,out	m	The matrix to modify.

11.17.4.15 MML_GDC_2D_MATRIX_API void utMat3x3Multiply (Mat3x3 dst, const Mat3x3 src1, const Mat3x3 src2)

Multiply 2 3x3-matrices. The resulting matrix represents dst = src1 ∗ src2.

Parameters

out	dst	The destination matrix.
in	src1	The first source matrix.
in	src2	The second source matrix.

11.17.4.16 MML_GDC_2D_MATRIX_API void utMat3x3RotX (Mat3x3 m, MM_FLOAT f)

Rotate a 3x3-matrix around the X-axis.

Parameters

in,out	m	The input/output matrix.
in	f	The rotation angle (in radians).

11.17.4.17 MML_GDC_2D_MATRIX_API void utMat3x3RotZ (Mat3x3 m, MM_FLOAT f)

Rotate a 3x3-matrix around the Z-axis.

Parameters

in,out	m	The input/output matrix.
in	f	The rotation angle (in radians).

11.17.4.18 MML_GDC_2D_MATRIX_API void utMat3x3Scale (Mat3x3 m, MM_FLOAT x, MM_FLOAT y)

Modify a 3x3-matrix to realize a scale operation. The resulting matrix represents m = m ∗ m_scale.

Parameters

in,out	m	The matrix to modify.
in	x	Scale factor in x direction.
in	y	Scale factor in y direction.

11.17.4.19 MML_GDC_2D_MATRIX_API void utMat3x3ToMat4x4 (Mat3x3 src, Mat4x4 dst)

Convert a 3x3-matrix to a 4x4-matrix.

Parameters

out	dst	The destination matrix.
in	src	The source matrix.

11.17.4.20 MML_GDC_2D_MATRIX_API void utMat3x3Translate (Mat3x3 m, MM_FLOAT x, MM_FLOAT y)

Modify a 3x3-matrix to realize a move operation. The resulting matrix represents m = m ∗ m_trans.

Parameters

in,out	m	The matrix to modify.
in	x	Move dimension in x direction.
in	y	Move dimension in y direction.

11.17.4.21 MML_GDC_2D_MATRIX_API void utMat3x3TranslatePre (Mat3x3 m, MM_FLOAT x, MM_FLOAT y)

Modify a 3x3-matrix by pre-multiplying a move matrix. The resulting matrix represents m = m_trans ∗ m.

Parameters

in,out	m	The matrix to modify.
in	x	Move dimension in x direction.
in	y	Move dimension in y direction.

11.17.4.22 MML_GDC_2D_MATRIX_API void utMat4x3CalcColMatrix (Mat4x3 dst, MM_FLOAT fContrast, MM_FLOAT fBrightness, MM_FLOAT fSaturation, MM_FLOAT fHue)

Calculate a color matrix with given parameters.

Parameters

out	dst	Destination color matrix. Previous matrix values will be overwritten.
in	fContrast	Set the contrast (color component amplification). A useful range for fContrast is 0.0 .. 2.0 with 1.0 stands for no contrast modifications and higher and lower values stand for amplification and attenuation.
in	fBrightness	Set the brightness (color component offset). A useful range for fBrightness is -1.0 .. 1.0 with 0.0 stands for no brightness modifications and higher and lower values stand for amplification and attenuation.
in	fSaturation	Set the color saturation. A useful range for fSaturation is 0.0 .. 2.0 with 1.0 stands for no saturation modifications and higher and lower values stand for amplification and attenuation. A value of 0.0 will result in a gray image.
in	fHue	Color modification in degrees. The useful range is 0 .. 360 where 0 and 360 run into a identical result which means no modification.

Note

- *The parameters are not checked concerning the range. A wrong value will result in a wrong image.*

11.17.4.23 MML_GDC_2D_MATRIX_API void utMat4x3Copy (Mat4x3 dst, const Mat4x3 src)

Copy the matrix content to a new one.

Parameters

out	dst	The destination matrix.
in	src	The source matrix.

11.17.4.24 MML_GDC_2D_MATRIX_API void utMat4x3LoadIdentity (Mat4x3 m)

Fill a 4x3-matrix with a unit matrix.

Parameters

in,out	m	The matrix to modify.

11.17.4.25 MML_GDC_2D_MATRIX_API void utMat4x3Multiply (Mat4x3 dst, const Mat4x3 src1, const Mat4x3 src2)

Multiply 2 matrices.

Parameters

out	dst	The destination matrix.
in	src1	The first source matrix.
in	src2	The second source matrix.

11.17.4.26 MML_GDC_2D_MATRIX_API void utMat4x4Copy (Mat4x4 dst, const Mat4x4 src)

Copy the content of a 4x4-matrix to a new one.

Parameters

out	dst	The destination matrix.
in	src	The source matrix.

11.17.4.27 MML_GDC_2D_MATRIX_API void utMat4x4GetXYZ (Mat4x4 m, float x, float y, float z, float * xout, float *yout, float * zout)

Calculate the target position for a given matrix and source position.

Parameters

in	m	The matrix.
in	x	Source x position.
in	y	Source y position.
in	z	Source z position.
out	xout	Pointer to the destination x position.
out	yout	Pointer to the destination y position.
out	zout	Pointer to the destination z position.

11.17.4.28 MML_GDC_2D_MATRIX_API void utMat4x4LoadIdentity (Mat4x4 m)

Fill a 4x4-matrix with a unit matrix.

Parameters

in,out	m	The matrix to modify.

11.17.4.29 MML_GDC_2D_MATRIX_API void utMat4x4Multiply (Mat4x4 dst, const Mat4x4 src1, const Mat4x4 src2)

Multiply 2 4x4-matrices. The resulting matrix represents dst = src1 * src2.

Parameters

out	dst	The destination matrix.
in	src1	The first source matrix.
in	src2	The second source matrix.

11.17.4.30 MML_GDC_2D_MATRIX_API void utMat4x4Perspective (Mat4x4 m, MM_FLOAT fovy, MM_FLOAT aspect, MM_FLOAT zNear, MM_FLOAT zFar)

Apply a perspective projection onto a 4x4-matrix.

Parameters

in,out	m	The input/output matrix.
in	fovy	The opening angle of the frustrum (in degrees).
in	aspect	The ratio of width/height.
in	zNear	The near distance.
in	zFar	The far distance.

11.17.4.31 MML_GDC_2D_MATRIX_API void utMat4x4RotX (Mat4x4 m, MM_FLOAT f)

Rotate a 4x4-matrix around the X-axis.

Parameters

in,out	m	The input/output matrix.
in	f	The rotation angle (in radians).

11.17.4.32 MML_GDC_2D_MATRIX_API void utMat4x4RotY (Mat4x4 m, MM_FLOAT f)

Rotate a 4x4-matrix around the Y-axis.

Parameters

in,out	m	The input/output matrix.
in	f	The rotation angle (in radians).

11.17.4.33 MML_GDC_2D_MATRIX_API void utMat4x4RotZ (Mat4x4 m, MM_FLOAT f)

Rotate a 4x4-matrix around the Z-axis.

Parameters

in,out	m	The input/output matrix.
in	f	The rotation angle (in radians).

11.17.4.34 MML_GDC_2D_MATRIX_API void utMat4x4Scale (Mat4x4 m, MM_FLOAT x, MM_FLOAT y, MM_FLOAT z)

Modify a 4x4-matrix to realize a scale operation. The resulting matrix represents m = m ∗ m_scale.

Parameters

in,out	m	The matrix to modify.
in	x	Scale factor in x direction.
in	y	Scale factor in y direction.
in	z	Scale factor in z direction.

11.17.4.35 MML_GDC_2D_MATRIX_API void utMat4x4ToMat3x2 (Mat4x4 src, Mat3x2 dst)

Convert a 4x4-matrix to a 3x2-matrix.

Parameters

out	dst	The destination matrix.
in	src	The source matrix.

11.17.4.36 MML_GDC_2D_MATRIX_API void utMat4x4ToMat3x3 (Mat4x4 src, Mat3x3 dst)

Convert a 4x4-matrix to a 3x3-matrix.

Parameters

out	dst	The destination matrix.
in	src	The source matrix.

11.17.4.37 MML_GDC_2D_MATRIX_API void utMat4x4Translate (Mat4x4 m, MM_FLOAT x, MM_FLOAT y, MM_FLOAT z)

Modify a 4x4-matrix to realize a move operation. The resulting matrix represents m = m ∗ m_trans.

Parameters

in,out	m	The matrix to modify.
in	x	Move dimension in x direction.
in	y	Move dimension in y direction.
in	z	Move dimension in z direction.

11.17.4.38 MML_GDC_2D_MATRIX_API void utMat5x4LoadIdentity (Mat5x4 m)

Fill a 5x4-matrix with a unit matrix.

Parameters

in,out	m	The matrix to modify.

11.18 Utilities for the compatibility with other drivers

Enumerations

- enum UTIL_VRAM_CONFIG {
  UTIL_VRAM_CONFIG_VRAM_ONLY = 0x1U,
  UTIL_VRAM_CONFIG_SDRAM_ONLY = 0x2U,
  UTIL_VRAM_CONFIG_VRAM_PREFERRED = 0x3U
  }

Functions

- MM_ERROR mmlGdcSmGenSurfaceObjects (MM_U32 uCnt, MML_GDC_SURFACE ∗pSurfaces)
- MM_ERROR mmlGdcSmDeleteSurfaceObjects (MM_U32 uCnt, MML_GDC_SURFACE ∗pSurfaces)

- MM_ERROR mmlGdcPeGenContext (MML_GDC_PE_CONTEXT *pPectx)
- void mmlGdcPeDeleteContext (MML_GDC_PE_CONTEXT pectx)
- void * mmlOsLibcMalloc (size_t _Size)
- void mmlOsLibcFree (void *_Memory)
- MM_ERROR mmlGdcVideoConfig (UTIL_VRAM_CONFIG config)
- void * mmlGdcVideoAlloc (MM_U32 size, MM_U32 alignment, MM_ADDR *pAddr)
- void mmlGdcVideoFree (void *addr)
- MM_ERROR mmlGdcVideoGetSize (MM_U32 *size)
- MM_ERROR mmlGdcVideoGetFreeTotal (MM_U32 *size)
- MM_ERROR mmlGdcVideoGetLargestBlock (MM_U32 *size)
- MM_ERROR mmlGdcSyncCreate (MM_U32 uCnt, MML_GDC_SYNC *pSyncObjects)
- MM_ERROR mmlGdcSyncDelete (MM_U32 uCnt, MML_GDC_SYNC *pSyncObjects)

11.18.1 Detailed Description

The functions of this group are used in some samples to make the application code identical to other 2D core based devices. For instance the mmlGdcSmGenSurfaceObjects() function is not available in this driver API for this hardware because system memory allocation is not allowed.

11.18.2 Enumeration Type Documentation

11.18.2.1 enum UTIL_VRAM_CONFIG

Configuration of video memory manager. This defines the memory region, where VideoAlloc shall allocate memory.

Enumerator

UTIL_VRAM_CONFIG_VRAM_ONLY Allocate memory from VRAM.

UTIL_VRAM_CONFIG_SDRAM_ONLY Allocate memory from SDRAM.

UTIL_VRAM_CONFIG_VRAM_PREFERRED Try to allocate memory from VRAM. If this fails, allocate memory from SDRAM.

11.18.3 Function Documentation

11.18.3.1 void mmlGdcPeDeleteContext (MML_GDC_PE_CONTEXT pectx)

mmlGdcPeDeleteContext deletes a context.

Parameters

in	pectx	The MML_GDC_PE_CONTEXT object

Return values

MML_OK	On success. Otherwise the related error code.

11.18.3.2 MM_ERROR mmlGdcPeGenContext (MML_GDC_PE_CONTEXT * pPectx)

mmlGdcPeGenContext creates pixel engine context.

Note:

- *Each function with a MML_GDC_PE_CONTEXT as parameter requires a previous call of mmlGdcPeGenContext for this context.*
- *The context will be initialized with default values. Please check the related property change functions to check the default values.*

Parameters

in,out	pPectx	Pointer to get the MML_GDC_PE_CONTEXT object.

Return values

MML_OK	On success. Otherwise the related error code.

11.18.3.3 MM_ERROR mmlGdcSmDeleteSurfaceObjects (MM_U32 uCnt, MML_GDC_SURFACE * pSurfaces)

Deletes a list of surface objects.

Note:

- *This function deletes the state-containing surface object.*

Parameters

in	uCnt	The number of surfaces to delete.
in	pSurfaces	The array of surfaces to delete.

Return values

MML_OK	On success.
MML_ERR_GDC_SURF_INVALID_PARAMETER	If NULL pointer is given for pSurfaces.

11.18.3.4 MM_ERROR mmlGdcSmGenSurfaceObjects (MM_U32 uCnt, MML_GDC_SURFACE * pSurfaces)

Creates uCnt empty surface objects, returning their names.

Note:

- *This function only instantiates empty surface objects. Before being used, they must be initialized by a function like mmlGdcSmAssignBuffer or the application must set their parameters manually.*

Parameters

in	uCnt	The number of surface objects to create.
out	pSurfaces	The output array for the returned names.

Return values

MML_OK	On success.
MML_ERR_GDC_SURF_INVALID_PARAMETER	If NULL pointer is given for pSurfaces.
MML_ERR_GDC_SURF_OUT_OF_SPACE	If not enough memory to create the surface object.

11.18.3.5 MM_ERROR mmlGdcSyncCreate (MM_U32 uCnt, MML_GDC_SYNC * pSyncObjects)

Creates cnt empty sync objects, returning their names.

Parameters

in	uCnt	Number of sync objects to create.
out	pSyncObjects	Output array for the returned names.

Return values

MML_OK	Success.
MML_ERR_GDC_SYNC_INVALID_PARAMETER	Invalid parameter.
MML_ERR_GDC_SYNC_OUT_OF_MEMORY	Out of memory.

11.18.3.6 MM_ERROR mmlGdcSyncDelete (MM_U32 uCnt, MML_GDC_SYNC * pSyncObjects)

Deletes a list of sync objects.

Parameters

in	uCnt	Number of sync objects to delete.
in	pSyncObjects	Array of sync objects to delete.

Return values

MML_OK	Success.
MML_ERR_GDC_SYNC_INVALID_PARAMETER	Invalid parameter.

11.18.3.7 void* mmlGdcVideoAlloc (MM_U32 size, MM_U32 alignment, MM_ADDR * pAddr)

Allocate a contiguous block of video memory.

Parameters

in	size	Amount of memory to be allocated in bytes.
in	alignment	Alignment to use for the allocation.
out	pAddr	If non-NULL, a pointer to a variable to receive the physical address of the memory block on success.

Returns

NULL on failure, or the virtual address of the allocated memory.

11.18.3.8 MM_ERROR mmlGdcVideoConfig (UTIL_VRAM_CONFIG config)

Configuration of video memory manager. This defines the memory region, where VideoAlloc shall allocate memory.

Parameters

in	config	Video memory manager configuration (default: UTIL_VRAM_CONFIG_VRAM_PREFERRED).

Returns

MML_ERR_MMAN_INVALID_PARAMETER, If illegal value for config is given, MML_OK otherwise.

11.18.3.9 void mmlGdcVideoFree (void * addr)

Free video memory allocated by mmlGdcVideoAlloc.

Parameters

in	addr	Virtual address previously returned from mmlGdcVideoAlloc.

11.18.3.10 MM_ERROR mmlGdcVideoGetFreeTotal (MM_U32 * size)

Retrieve the total amount of free video memory. Depending on the configuration (see mmlGdcVideoConfig), this refers to VRAM, SDRAM or both.

Parameters

out	size	Parameter to receive the query result [not NULL].

Return values

MML_OK	Normal termination.
MML_ERR_MMAN_INVALID_PARAMETER	An unexpected internal error occurred.

11.18.3.11 MM_ERROR mmlGdcVideoGetLargestBlock (MM_U32 * size)

Retrieve the size of the largest contiguous block of free video memory. Depending on the configuration (see mmlGdcVideoConfig), this refers to VRAM, SDRAM or both.

Parameters

out	size	Parameter to receive the query result [not NULL].

Return values

MML_OK	Normal termination.
MML_ERR_MMAN_INVALID_PARAMETER	An unexpected internal error occurred.

11.18.3.12 MM_ERROR mmlGdcVideoGetSize (MM_U32 * size)

Retrieve the size of video memory heap. Depending on the configuration (see mmlGdcVideoConfig), this is the size of VRAM, SDRAM or both.

Parameters

out	size	Parameter to receive the query result [not NULL].

Return values

MML_OK	Normal termination.
MML_ERR_MMAN_INVALID_PARAMETER	An unexpected internal error occurred.

11.18.3.13 void mmlOsLibcFree (void * _Memory)

Implements the standard C Library function free().

Parameters

in	_Memory	Virtual address previously returned from mmlOsLibcMalloc.

11.18.3.14 void* mmlOsLibcMalloc (size_t _Size)

Implements the standard C Library function malloc().

Parameters

in	_Size	Amount of memory to be allocated in bytes.

Returns

NULL on failure, or the virtual address of the allocated memory.

11.19 Utilities for the Surface Management

Macros

- #define UTIL_SUCCESS(rc, execute)
- #define UTIL_ERR_OUT_OF_MEMORY MM_ERRCODE(0x31000001)

Functions

- MM_ERROR utSurfReadBitmap (MML_GDC_SURFACE surface, void **pImage, MM_U32 *baseAddr, MM_U32 *clutAddr)
- MM_ERROR utSurfLoadBitmap (MML_GDC_SURFACE surface, const void *pImage, MM_BOOL bCopyToVRAM)
- MM_S32 utSurfWidth (MML_GDC_SURFACE surf)
- MM_S32 utSurfHeight (MML_GDC_SURFACE surf)
- MM_ERROR utSurfCreateBuffer (MML_GDC_SURFACE surf, MM_U32 w, MM_U32 h, MML_GDC_SURF_FORMAT eFormat)
- void utSurfDeleteBuffer (MML_GDC_SURFACE surf)
- MM_ERROR utSurfGetPixel (MML_GDC_SURFACE src, MM_U32 x, MM_U32 y, MM_U08 *r, MM_U08 *g, MM_U08 *b, MM_U08 *a)
- MM_ERROR utSurfSetPixel (MML_GDC_SURFACE src, MM_U32 x, MM_U32 y, MM_U08 r, MM_U08 g, MM_U08 b, MM_U08 a)

11.19.1 Detailed Description

This utility block realizes some helper functions related to the surface manager API of the 2D core graphics driver. The following code allocates an 16bpp image buffer in the VRAM and initializes a surface object. Afterwards it fills the surface with generated pixel data.

```
void CreatePattern(MML_GDC_SURFACE surf, MM_U32 width, MM_U32 height)
{
    MM_U32 x;
    MM_U32 y;
    MM_U32 red;
    MM_U32 green;
    MM_U32 blue;
    MM_U32 alpha;
    mmlGdcSmResetSurfaceObject(surf);
    utSurfCreateBuffer(surf, width, height, MML_GDC_SURF_FORMAT_R4G4B4A4);
    for (x = 0; x < width; x++)
    {
        for (y = 0; y < height; y++)
        {
            red = 255 - 255 * x / width;
            green = 255 * x / width;
            blue = 255 * y / width;
            alpha = 255 - 255 * y / width;
            utSurfSetPixel(surf, x, y, red, green, blue, alpha);
```

```
        }
    }
}
```

11.19.2 Macro Definition Documentation

11.19.2.1 #define UTIL_ERR_OUT_OF_MEMORY MM_ERRCODE(0x31000001)

Out of memory

11.19.2.2 #define UTIL_SUCCESS(rc, execute)

Value:

```
do ¥
{ ¥
    if ((rc) ==  MML_OK) ¥
    { ¥
        rc = (execute); ¥
        if (rc != MML_OK)
            printf("Error %x in %s (%s line %d)¥n", (int)rc, FUNCTION, FILE, LINE);¥
    } ¥
} while (0)
```

This macro avoids execution if the previous instruction failed.

11.19.3 Function Documentation

11.19.3.1 MM_ERROR utSurfCreateBuffer (MML_GDC_SURFACE surf, MM_U32 w, MM_U32 h, MML_GDC_SURF_FORMAT eFormat)

Create a buffer with the given dimension and color format and set the related surface object properties. Please note the surface object must be created before.

Parameters

in	surf	The surface object
in	w	Width of the surface
in	h	Height of the surface
in	eFormat	The requested color format

Return values

MML_OK	on success, otherwise the related error code.

11.19.3.2 void utSurfDeleteBuffer (MML_GDC_SURFACE surf)

Delete the surface buffers.

Parameters

in	surf	The surface object

11.19.3.3 MM_ERROR utSurfGetPixel (MML_GDC_SURFACE src, MM_U32 x, MM_U32 y, MM_U08 * r, MM_U08 * g, MM_U08 * b, MM_U08 * a)

Get the r, g, b, a pixel data of a surface at position x, y. The pixel with the coordinates 0, 0 is the upper, left pixel analoge to the the memory organisation of the memory buffer. Please note that this is different to the glReadPixels specification. Please note not all possible surface formates are supported.

Parameters

in	src	The surface object
in	x	x positon of the pixel
in	y	y positon of the pixel
in	r	Pointer to get the red value
in	g	Pointer to get the green value
in	b	Pointer to get the blue value
in	a	Pointer to get the alpha value

Return values

MML_OK	on success, otherwise the related error code.

11.19.3.4 MM_S32 utSurfHeight (MML_GDC_SURFACE surf)

Return the height of a given surface object

Parameters

in	surf	The surface object

Return values

Height	of the surface.

11.19.3.5 MM_ERROR utSurfLoadBitmap (MML_GDC_SURFACE surface, const void * pImage, MM_BOOL bCopyToVRAM)

Read a bitmap structure, set the related surface attributes, allocate the required memory for pixel and CLUT buffer and copy the related data. Please note the surface object must be created before.

Parameters

in	surface	The surface object
in	pImage	pointer to the image
in	bCopyToVRAM	MM_TRUE if the bitmap and color look up table memory should be copied into VRAM.

Return values

MML_OK	on success, otherwise the related error code.

11.19.3.6 MM_ERROR utSurfReadBitmap (MML_GDC_SURFACE surface, void ** pImage, MM_U32 * baseAddr, MM_U32 * clutAddr)

Read a bitmap structure in memory and set the related surface attributes Please note the surface object must be created before. No memory will be allocated in this function. The pImage pointer will be increased by the size of the whole image so it points to the next image object if further images are in the memory block.

Parameters

in	surface	The surface object
in	pImage	address of the pointer to the image
in	baseAddr	Pointer to get color buffer virtual base address
in	clutAddr	Pointer to get clut buffer virtual base address

Return values

MML_OK	on success, otherwise the related error code.

11.19.3.7 MM_ERROR utSurfSetPixel (MML_GDC_SURFACE src, MM_U32 x, MM_U32 y, MM_U08 r, MM_U08 g, MM_U08 b, MM_U08 a)

Set the r, g, b, a pixel data of a surface at position x, y The pixel with the coordinates 0, 0 is the upper, left pixel analoge to the the memory organisation of the memory buffer. Please note that this is different to the glReadPixels specification. Please note not all possible surface formates are supported.

Parameters

in	src	The surface object
in	x	x positon of the pixel
in	y	y positon of the pixel
in	r	New red value
in	g	New green value
in	b	New blue value
in	a	New alpha value

Return values

MML_OK	on success, otherwise the related error code.

11.19.3.8 MM_S32 utSurfWidth (MML_GDC_SURFACE surf)

Return the width of a given surface object

Parameters

in	surf	The surface object

Return values

Width	of the surface.

11.20 Utilities for the compression

Modules

- Utilities for RLA (run length adaptive compression)
- Utilities for RLC (run length compression)

Functions

- MM_ERROR utSurfCompress (MML_GDC_SURFACE surf, MML_GDC_SURF_COMP mode)

11.20.1 Detailed Description

This group contains sample helper functions for surface compression. It shows how the surface parameters must be used with the Utilities for RLA (run length adaptive compression) and Utilities for RLC (run length compression) utilities.

11.20.2 Function Documentation

11.20.2.1 MM_ERROR utSurfCompress (MML_GDC_SURFACE surf, MML_GDC_SURF_COMP mode)

Compress a surface buffer.

Note:

- *This function shows how images can be compresses to reduce the memory usage. The compressed images can be used as source surfaces for blit and display operations. A real application will probably not use this function but only load uses such compressed buffers in an application. Moreover this function may fail for large images because the system memory is not sufficient.*

Parameters

in,out	surf	The surface object describing an uncompressed image buffer. If the compression was successful the surface object describes the new compressed buffer.
in	mode	The requested compression mode.

Return values

MML_OK	On success.
MML_ERR	If the requested compression is not possible.

11.21 Utilities for RLA (run length adaptive compression)

Data Structures

- class RLAD

11.21.1 Detailed Description

The code for this group can be used to create compressed buffers of the type MML_GDC_SURF_COMP_RLA, MML_GDC_SURF_COMP_RLAD and ::MML_GDC_SURF_COMP_RLAD_UNIFORM.

Note:

- *The header and the source code for this functions are included in the utility block delivered with the driver although it is not recommended to compress an image with the CPU on the target system. However, if required this part can used to create compression utilities for different platforms.*

11.22 Utilities for RLC (run length compression)

Functions

- MM_U32 utRldEncode (MM_U32 *pixeldata, MM_U32 unWidth, MM_U32 unHeight, MM_U32 strideBytes, MM_U32 dataBpp, MM_U32 *rld, MM_U32 rldCount)

11.22.1 Detailed Description

This group defines function to create run length compression streams.

Note:

- *The header and the source code for this functions are included in the utility block delivered with the driver although it is not recommended to compress an image with the CPU on the target system. However, if required this part can used to create compression utilities for different platforms.*

11.22.2 Function Documentation

11.22.2.1 MM_U32 utRldEncode (MM_U32 * pixeldata, MM_U32 unWidth, MM_U32 unHeight, MM_U32 strideBytes, MM_U32 dataBpp, MM_U32 * rld, MM_U32 rldCount)

Encode pixel data to RLD bit stream.

Parameters

in	pixeldata	Pixel data.
in	unWidth	Width of the image.
in	unHeight	Height of the image.
in	strideBytes	Number of bytes required for one line.
in	dataBpp	Bits per pixel (1, 2, 4, 8, 16, 24, 32).
out	rld	RLD bit stream.
in	rldCount	Maximum number of RLD words.

Return values

Required	number of RLD words. This number may me larger than rldCount, in which case only rldCount words are actually written.

Note:

- *The RLD bit stream is filled up with zero bits at the end, for alignment with word boundaries. RLD will ignore the fill bits since the expected data size is provided as a parameter for decoding.*

11.23 Util class collection

Modules

- CCtx
- CDevice
- CDisplay
- CMenu
- CSurface
- CWindow

11.23.1 Detailed Description

The util class collection defines some classes to abstract low level driver functionality. All these classes are defined as header files only.

11.24 CCtx

Data Structures

- class CCtx

11.24.1 Detailed Description

The class CCtx is a simple abstraction of a MML_GDC_PE_CONTEXT object. The application can use an object of this calls direct for blitting because the constructor takes over the initialization of the context. After a call of OpenDrawCtx this context can be also used for drawing.

11.25 CDevice

Data Structures

- class CDevice

11.25.1 Detailed Description

The Class CDevice is responsible to initialize the 2D core driver and util part in the Open() function and also allocates and assigns command sequencer fifo. It is required that this Open() function is called before using any other util class functions and the application must also ensure that the device destruction is called as the last instruction of an application. Only one object of the CDevice call is allowed in a program.

11.26 CDisplay

Data Structures

- class CDisplay

11.26.1 Detailed Description

The class CDisplay abstracts a MML_GDC_DISPLAY object and adds some helper functions.

11.27 CMenu

Data Structures

- class CMenu

11.27.1 Detailed Description

The classes in this group realize a simple menu. It can be used with an 2D core display layer for demo applications to allow selections, switches and similar operations with a minimum keys.

Sample code:

```
CMenu menu;

menu.Open(display, 0, 0, 320, 240, MML_GDC_DISP_LAYER_4,
MML_GDC_DISP_SUB_LAYER_DEFAULT,    0, MML_GDC_DISP_BLEND_SOURCE_ALPHA |
MML_GDC_DISP_BLEND_SOURCE_MULTIPLY_ALPHA);
menu.InitMenu(Font_ttf, sizeof(Font_ttf), 16);

menu.Insert(0, MENU_LL, CMenu::MENU_FLAG_CHECKBOX , L"This is a check box menu");
menu.Insert(MENU_LL, MENU_FONT, 0, L"Font");
menu.Insert(MENU_FONT + 0, MENU_FONT + 1, CMenu::MENU_FLAG_POPUP |
CMenu::MENU_FLAG_RADIO | CMenu::MENU_FLAG_ISCHECKED,    L"Font 1");
menu.Insert(MENU_FONT + i, MENU_FONT + i + 1, CMenu::MENU_FLAG_RADIO , L"Font 2");

while(bRunning)
{
    key = menu.HandleKey(GetLastKeyStroke());
    switch(key()
    {
        ...
    }
    menu.Draw();

    //draw other things
    ...
}
```

11.28 CSurface

Data Structures

- class CSurface< NUM_BUFFERS >

Functions

- CSurface ()
- void Init ()
- MM_ERROR Delete ()
- virtual MM_ERROR CreateBuffer (const MM_U32 width, const MM_U32 height, const MML_GDC_SURF_FORMAT format=MML_GDC_SURF_FORMAT_R8G8B8A8, MM_U32 MaxSize=0)
- virtual MM_ERROR CreateBuffer (const MM_U32 width, const MM_U32 height, MM_U32 bit_red, MM_U32 bit_green, MM_U32 bit_blue, MM_U32 bit_alpha)
- virtual MM_ERROR CreateGrayBuffer (const MM_U32 width, const MM_U32 height, MM_U32 bit_color, MM_U32 bit_alpha)
- virtual MM_ERROR SurfLoadBitmap (const void *pImage, MM_BOOL bCopyToVRAM=MM_FALSE)
- virtual MM_ERROR Copy (MML_GDC_SURFACE surface)

11.28.1 Detailed Description

The class CSurface is a abstraction of one or more MML_GDC_SURFACE objects depending on the NUM_BUFFERS definition. The constructor takes over the surface object initialization. To use the surface for blit or display operations it is typically required to allocate VRAM or to assign a static resource from FLASH memory. The required functions are part of these class.

If the NUM_BUFFERS is 2 (or more), the CSurface object can be used for multi buffer rendering. The Swap member function can be used to toggle between foreground and background buffer. The GetSurface, GetHandle and []operator will always return the foreground buffer.

11.28.2 Function Documentation

11.28.2.1 MM_ERROR Copy (MML_GDC_SURFACE surface) [virtual]

Copy the surface object.

Note:

- *This function copies the properties only. Not the surface content of surface object.*

Parameters

in	surface	Surface that should be copied.

Return values

MML_OK	on success. Otherwise the related error code.

11.28.2.2 MM_ERROR CreateBuffer (const MM_U32 width, const MM_U32 height, const MML_GDC_SURF_FORMAT format = MML_GDC_SURF_FORMAT_R8G8B8A8, MM_U32 MaxSize = 0) [virtual]

The CreateBuffer function can be used setup the member surface object(s) with allocated VRAM.

Parameters

in	width	Defines the with of the surface(s).
in	height	Defines the height of the surface(s).
in	format	Defines the color format of the surface(s).
in	MaxSize	Experimental: Defines the maximum size for the buffers. O: No limitation, the required buffer will be allocated. Size in bytes: the buffer will be created with ::MML_GDC_SURF_COMP_RLAD_UNIFORM parameter and the allocated buffer size will be smaller or equal MaxSize.

Return values

MML_OK	on success. Otherwise the related error code.

11.28.2.3 MM_ERROR CreateBuffer (const MM_U32 width, const MM_U32 height, MM_U32 bit_red, MM_U32 bit_green, MM_U32 bit_blue, MM_U32 bit_alpha) [virtual]

The CreateBuffer function can be used setup the member surface object(s) with allocated VRAM.

Parameters

in	width	Defines the with of the surface(s).
in	height	Defines the height of the surface(s).
in	bit_red	Defines the bits for the red channel of the surface(s).
in	bit_green	Defines the bits for the green channel of the surface(s).
in	bit_blue	Defines the bits for the blue channel of the surface(s).
in	bit_alpha	Defines the bits for the alpha channel of the surface(s).

Return values

MML_OK	on success. Otherwise the related error code.

11.28.2.4 MM_ERROR CreateGrayBuffer (const MM_U32 width, const MM_U32 height, MM_U32 bit_color, MM_U32 bit_alpha) [virtual]

The CreateBuffer function can be used setup the member surface object(s) with allocated VRAM.

Parameters

in	width	Defines the with of the surface(s).
in	height	Defines the height of the surface(s).
in	bit_color	Defines the common bits for the red, green and blue channel of the surface(s).
in	bit_alpha	Defines the bits for the alpha channel of the surface(s).

Return values

MML_OK	on success. Otherwise the related error code.

11.28.2.5 CSurface ( )

Class CSurface constructor.

11.28.2.6 MM_ERROR Delete ( )

The Delete function can be used to free up allocated memory (if any). This function will be called in destructor too.

Return values

MML_OK	on success. Otherwise the related error code.

11.28.2.7 void Init ( )

Class CSurface init function. This functions does exactly the same as the constructor. It is needed for some compilers (at the moment ghs and gnu), not running the constructors of global classes before main. This function can be called from main as a workarroung.

11.28.2.8 MM_ERROR SurfLoadBitmap (const void * pImage, MM_BOOL bCopyToVRAM = MM_FALSE) [virtual]

The CreateBuffer function uses the utSurfLoadBitmap function to initialize the current surface object.

Parameters

in	pImage	Pointer to the image
in	bCopyToVRAM	MM_TRUE if the bitmap and color look up table memory should be copied into VRAM.

Return values

MML_OK	on success. Otherwise the related error code.

11.29 CWindow

Data Structures

- class CWindow
- class CSurfaceWindow< NUM_BUFFERS >
- class CStaticSurfaceWindow

11.29.1 Detailed Description

The class CWindow can be used to open a window with the 2D core driver. The header file contains different derived classed for different use cases. For instance a CSurfaceWindow<2> object can be used to manage a double buffered render target that will be displayed as a window layer on the connected panel. A CStaticSurfaceWindow object can be used as a static background layer or to display a static icon as foreground window. The typical use or these CWindows object will be shown in the following sample:

```
void main()
{
```

```
    CDevice                 device;
    CDisplay                display;
    CStaticSurfaceWindow wndBg;
    CSurfaceWindow<2>       wndRender;

    //open device
    device.Open();
    // open display
    display.Open(ScreenWidth, ScreenHeight);
    // open a background window and assign an image
    wndBg.Open(display, background_image);

    // open a foreground window with alpha blending
    wndRender.Open(display, 0, 0, ScreenWidth, ScreenHeight, MML_GDC_DISP_LAYER_1,
MML_GDC_DISP_SUB_LAYER_DEFAULT, 0, MML_GDC_DISP_BLEND_SOURCE_ALPHA);
    //create a (double) buffer for the window
    wndRender.CreateBuffer();
    while (draw)
    {
        // render something to wndRender.m_surface
        ...

        //swap the buffers
        wndRender.Swap();
    }
}
```

12. Data Structure Documentation

12.1 RLAD::BitStream Class Reference

#include <ut_class_rlad.h>

Public Member Functions

- BitStream (bool big_endian=false)
- unsigned Size () const
- bool IsBigEndian () const
- void Push (unsigned bits, unsigned data)
- void Clear ()
- unsigned Read (unsigned bits, bool *err=0)
- void ResetRead ()

Friends

- class RLAD

12.1.1 Detailed Description

The class BitStream is used to store the compressed image

12.1.2 Constructor & Destructor Documentation

12.1.2.1 BitStream (bool big_endian = false) [inline]

Constructor

Parameters

in	big_endian	Set true if system is big_endian

12.1.3 Member Function Documentation

12.1.3.1 void Clear ( )

Reset stream

12.1.3.2 bool IsBigEndian ( ) const [inline]

Return values

Return	true if BigEndian

12.1.3.3 void Push (unsigned bits, unsigned data)

Push bits to the compressed stream

Parameters

in	bits	Number of bits in data
in	data	Data to push

12.1.3.4 unsigned Read (unsigned bits, bool * err = 0)

Read bits from the compressed stream

Parameters

in	bits	Number of bits to read
in,out	err	Will be set to true if error occurs

Return values

Read	data

12.1.3.5 void ResetRead ( )

reset Read operation to begin

12.1.3.6 unsigned Size ( ) const

Size in bits

The documentation for this class was generated from the following file:

- ut_class_rlad.h

12.2 CCtx Class Reference

#include <ut_class_ctx.h>

Public Member Functions

- CCtx ()
- ~CCtx ()
- void Init ()
- void Reset ()
- MML_GDC_PE_CONTEXT GetHandle ()
- operator MML_GDC_PE_CONTEXT ()

12.2.1 Detailed Description

Class CCtx see CCtx.

12.2.2 Constructor & Destructor Documentation

12.2.2.1 CCtx ( ) [inline]

Class CCtx constructor.

12.2.2.2 ~CCtx ( ) [inline]

Class CCtx destructor.

12.2.3 Member Function Documentation

12.2.3.1 MML_GDC_PE_CONTEXT GetHandle ( ) [inline]

Return the MML_GDC_PE_CONTEXT object

12.2.3.2 void Init ( ) [inline]

Initialize context variables.

12.2.3.3 operator MML_GDC_PE_CONTEXT ( ) [inline]

Return the MML_GDC_PE_CONTEXT object for direct use with 2D core driver API calls

12.2.3.4 void Reset ( ) [inline]

Reset the draw buffer

The documentation for this class was generated from the following file:

- ut_class_ctx.h

12.3 CDevice Class Reference

#include <ut_class_device.h>

Public Member Functions

- CDevice ()
- ~CDevice ()
- MM_ERROR Open (MM_U32 uCmdSeqSize=0)
- MM_ERROR Close ()
- MM_BOOL IsOpen ()

12.3.1 Detailed Description

Class CDevice.

12.3.2 Constructor & Destructor Documentation

12.3.2.1 CDevice ( ) [inline]

Class CDevice constructor.

12.3.2.2 ~CDevice ( ) [inline]

Class CDevice destructor.

12.3.3 Member Function Documentation

12.3.3.1 MM_ERROR Close ( ) [inline]

Close the device (will be called from destructor).

Return values

MML_OK	On success. Otherwise the related error code or MML_ERR.

12.3.3.2 MM_BOOL IsOpen ( ) [inline]

Can be used to check the status.

Return values

MM_TRUE	If Open was successfully called otherwise MM_FALSE.

12.3.3.3 MM_ERROR Open (MM_U32 uCmdSeqSize = 0) [inline]

Open the device .

Parameters

in	uCmdSeqSize	Defines the size that will be allocated for the command sequencer.

Return values

MML_OK	On success. Otherwise the related error code or MML_ERR.

The documentation for this class was generated from the following file:

- ut_class_device.h

12.4 CDisplay Class Reference

#include <ut_class_display.h>

Public Member Functions

- MM_ERROR Open (unsigned int nWidth=0, unsigned int nHeight=0, MML_GDC_DISP_CONTROLLER display=MML_GDC_DISP_CONTROLLER_0)
- MM_ERROR Close ()
- virtual MM_ERROR SetBgColor (MM_U32 color)
- MM_U32 GetWidth ()
- MM_U32 GetHeight ()
- MML_GDC_DISP_CONTROLLER GetDisplayController ()
- MML_GDC_DISPLAY GetHandle ()
- operator MML_GDC_DISPLAY ()

12.4.1 Detailed Description

Class CDisplay (see CDisplay).

12.4.2 Member Function Documentation

12.4.2.1 MM_ERROR Close ( ) [inline]

Close the display controller.

Return values

MML_OK	On success. Otherwise the related error code or MML_ERR.

12.4.2.2 MML_GDC_DISP_CONTROLLER GetDisplayController ( ) [inline]

Return values

Return	the used display controller id.

12.4.2.3 MML_GDC_DISPLAY GetHandle ( ) [inline]

Return values

Return	the used MML_GDC_DISPLAY object.

12.4.2.4 MM_U32 GetHeight ( ) [inline]

Return values

Return	the height of the panel.

12.4.2.5 MM_U32 GetWidth ( ) [inline]

Return values

Return	the width of the panel.

12.4.2.6 MM_ERROR Open (unsigned int nWidth = 0, unsigned int nHeight = 0, MML_GDC_DISP_CONTROLLER display = MML_GDC_DISP_CONTROLLER_0) [inline]

Open / initialize the 2D core display controller.

Parameters

in	nWidth	Number of horizontal pixel.
in	nHeight	Number of vertical pixel.
in	display	ID of display controller.

Return values

MML_OK	On success. Otherwise the related error code or MML_ERR.

12.4.2.7 operator MML_GDC_DISPLAY ( ) [inline]

Return values

Return	the used MML_GDC_DISPLAY object.

12.4.2.8 virtual MM_ERROR SetBgColor (MM_U32 color) [inline], [virtual]

Change the background color and applay changes with commit.

Parameters

in	color	see MML_GDC_DISP_ATTR_BACKGROUND_COLOR

Return values

MML_OK	On success. Otherwise the related error code or MML_ERR.

The documentation for this class was generated from the following file:

- ut_class_display.h

12.5 CMenu Class Reference

#include <ut_class_menu.h>

Public Types

- enum MENU_FLAG {
  MENU_FLAG_POPUP = 1,
  MENU_FLAG_CHECKBOX = CMenuItem::MENU_ITEM_FLAG_CHECKBOX,
  MENU_FLAG_RADIO = CMenuItem::MENU_ITEM_FLAG_RADIO,
  MENU_FLAG_ISCHECKED = CMenuItem::MENU_ITEM_FLAG_ISCHECKED
  }
- enum MENU_KEYS {
  MENU_KEY_ENTER = 0x10000000,
  MENU_KEY_UP = 0x10000001,
  MENU_KEY_DOWN = 0x10000002,
  MENU_KEY_LEFT = 0x10000003,
  MENU_KEY_RIGHT = 0x10000004
  }

Public Member Functions

- virtual MM_ERROR SetText (const wchar_t *pszString)
- MM_ERROR InitMenu (const void *Font, int size_of_font, int font_height, MML_GDC_PE_CONTEXT draw_ctx=0)
- virtual MM_ERROR Close ()
- MM_ERROR Insert (MM_U32 old_id, MM_U32 id, MM_U32 flags, const wchar_t *pszString)
- CMenuItem * Find (MM_U32 id, CMenuItem *pBase)
- CMenuItem * FindSelected (CMenuItem *pBase)
- CMenuItem * FindNext (CMenuItem *pBase, CMenuItem *pSearch, CMenuItem::MENU_ITEM_FIND find)
- int HandleKey (MM_U32 key)
- MM_U32 GetDefautItemHeight ()
- MM_ERROR Draw ()

12.5.1 Detailed Description

Class CMenu (see CMenu)

12.5.2 Member Enumeration Documentation

12.5.2.1 enum MENU_FLAG

Define some menu types and states.

Enumerator

MENU_FLAG_POPUP Popup menu entry.

MENU_FLAG_CHECKBOX Menu item with check box.

MENU_FLAG_RADIO Menu item with radio button.

MENU_FLAG_ISCHECKED Menu item is checked.

12.5.2.2 enum MENU_KEYS

Enumerator

MENU_KEY_ENTER Enter (select) key.

MENU_KEY_UP Up key.

MENU_KEY_DOWN Down key.

MENU_KEY_LEFT Left key.

MENU_KEY_RIGHT Right key.

12.5.3 Member Function Documentation

12.5.3.1 virtual MM_ERROR Close ( ) [inline], [virtual]

Close the menu

12.5.3.2 MM_ERROR Draw ( ) [inline]

Redraw the menu if required.

Return values

MML_OK	on success. Otherwise the related error code or MML_ERR.

12.5.3.3 CMenuItem* Find (MM_U32 id, CMenuItem * pBase) [inline]

Search a menu item in the menu.

Parameters

in	id	ID of the menu item to find.
in	pBase	Base menu item to start the search.

Return the menu item on success. Otherwise NULL.

12.5.3.4 CMenuItem* FindNext (CMenuItem * pBase, CMenuItem * pSearch, CMenuItem::MENU_ITEM_FIND find) [inline]

Search a menu item in the menu.

Parameters

in	pBase	Base menu item to start the search.
in	pSearch	Reverence menu item for the search.
in	find	Relation of the new item related to pSearch.

Return the menu item on success. Otherwise NULL.

12.5.3.5 CMenuItem* FindSelected (CMenuItem * pBase) [inline]

Find the current selected menu item.

Parameters

in	pBase	Base menu item to start the search.

Return the menu item on success. Otherwise NULL.

12.5.3.6 MM_U32 GetDefautItemHeight ( ) [inline]

Return values

Return	the height of one menu item.

12.5.3.7 int HandleKey (MM_U32 key) [inline]

Progress the key press input: for instance select the lower menu item if key down was pressed.

Parameters

in	key	

Return values

	The function return – The original key code if no action inside the menu was proceed. – 0 if an action was proceeded (e.g., selection changed). – Or the selected menu item id if the menu was opened and the enter key was pressed.

12.5.3.8 MM_ERROR InitMenu (const void * Font, int size_of_font, int font_height, MML_GDC_PE_CONTEXT draw_ctx = 0) [inline]

Initialize the menu.

Parameters

in	Font	Define the tt font for the menu (if size_of_font == 0 it defines the location in the file system; if size_of_font != 0, it is the pointer to the font buffer with a size of size_of_font).
in	size_of_font	See font parameter.
in	font_height	Define the requested font height in pixel.
in	draw_ctx	Initialized draw context objec.t

Return values

MML_OK	On success. Otherwise the related error code or MML_ERR.

12.5.3.9 MM_ERROR Insert (MM_U32 old_id, MM_U32 id, MM_U32 flags, const wchar_t * pszString) [inline]

Insert a menu item to the menu.

Parameters

in	old_id	ID of the parent menu item id or 0 if it is the root item (only one root item must be defined!).
in	id	ID of the inserted menu item.
in	flags	One or more "ored" MENU_FLAG's.
in	pszString	Menu item string.

Return values

MML_OK	On success. Otherwise the related error code or MML_ERR.

12.5.3.10 virtual MM_ERROR SetText (const wchar_t * pszString) [inline], [virtual]

Do not use this function!

The documentation for this class was generated from the following file:

- ut_class_menu.h

12.6 CMenuItem Class Reference

Public Types

- enum MENU_ITEM_FLAG {
  MENU_ITEM_FLAG_VISIBLE = 0x1000,
  MENU_ITEM_FLAG_FOCUS = 0x2000,
  MENU_ITEM_FLAG_CHECKBOX = 0x100,
  MENU_ITEM_FLAG_RADIO = 0x200,
  MENU_ITEM_FLAG_ISCHECKED = 0x400
  }
- enum MENU_ITEM_FIND {
  MENU_ITEM_FIND_UP,
  MENU_ITEM_FIND_DOWN,
  MENU_ITEM_FIND_TOP,
  MENU_ITEM_FIND_BOTTOM,
  MENU_ITEM_FIND_PARENT,
  MENU_ITEM_FIND_SELECTED
  }

Public Member Functions

- CMenuItem (MM_U32 id, MM_U32 flag, const wchar_t *pszString)
- const wchar_t * GetString ()

Data Fields

- CMenuItem * m_pSubItem
- CMenuItem * m_pNextItem
- MM_U32 m_id
- MM_U32 m_flag
- MM_U16 m_width
- MM_U08 m_nTextOffset
- MM_U08 m_height

Friends

- class CMenu

The documentation for this class was generated from the following file:

- ut_class_menu.h

12.7 CStaticSurfaceWindow Class Reference

#include <ut_class_window.h>

Public Member Functions

- virtual MM_ERROR Open (MML_GDC_DISPLAY display, const void *pImage, MM_BOOL bCopyToVRAM=MM_FALSE, MM_S32 x=0, MM_S32 y=0, MML_GDC_DISP_LAYER layerId=MML_GDC_DISP_LAYER_0, MML_GDC_DISP_SUB_LAYER sub_layerID=MML_GDC_DISP_SUB_LAYER_DEFAULT, MM_U32 blend_mode=0)

- virtual MM_ERROR Open (MML_GDC_DISPLAY display, MML_GDC_SURFACE sImage, MM_S32 x=0, MM_S32 y=0, MML_GDC_DISP_LAYER layerId=MML_GDC_DISP_LAYER_0, MML_GDC_DISP_SUB_LAYER sub_layerID=MML_GDC_DISP_SUB_LAYER_DEFAULT, MM_U32 blend_mode=0)

Additional Inherited Members

12.7.1 Detailed Description

The Class CStaticSurfaceWindow uses a 2D core window showing a static image like a background image or a for instance sign as foreground layer.

12.7.2 Member Function Documentation

12.7.2.1 virtual MM_ERROR Open (MML_GDC_DISPLAY display, const void * pImage, MM_BOOL bCopyToVRAM = MM_FALSE, MM_S32 x = 0, MM_S32 y = 0, MML_GDC_DISP_LAYER layerId = MML_GDC_DISP_LAYER_0, MML_GDC_DISP_SUB_LAYER sub_layerID = MML_GDC_DISP_SUB_LAYER_DEFAULT, MM_U32 blend_mode = 0) [inline], [virtual]

Open the window and show an image.

Parameters

in	display	A display object that will be used to open the window.
in	pImage	Pointer to an buffer array describing a 2D core pixel buffer (analog to utSurfLoadBitmap()).
in	bCopyToVRAM	If MM_TRUE the pImage will be copied to VRAM otherwise the display controller will read the pImage buffer direct.
in	x	X position offset of the upper left window corner relative to the display screen.
in	y	Y position offset of the upper left window corner relative to the display screen.
in	layerId	Layer ID of the window.
in	sub_layerID	Sub-Layer ID of the window.
in	blend_mode	Starting blend mode for the window see mmlGdcDispWinSetBlendMode().

Return values

MML_OK	on success. Otherwise the related error code or MML_ERR.

12.7.2.2 virtual MM_ERROR Open (MML_GDC_DISPLAY display, MML_GDC_SURFACE sImage, MM_S32 x = 0, MM_S32 y = 0, MML_GDC_DISP_LAYER layerId = MML_GDC_DISP_LAYER_0, MML_GDC_DISP_SUB_LAYER sub_layerID = MML_GDC_DISP_SUB_LAYER_DEFAULT, MM_U32 blend_mode = 0) [inline], [virtual]

Open the window and show an image.

Parameters

in	display	A display object that will be used to open the window.
in	sImage	The MML_GDC_SURFACE object to be displayed.
in	x	X position offset of the upper left window corner relative to the display screen.
in	y	Y position offset of the upper left window corner relative to the display screen.
in	layerId	Layer ID of the window.
in	sub_layerID	Sub-Layer ID of the window.
in	blend_mode	Starting blend mode for the window see mmlGdcDispWinSetBlendMode().

Return values

MML_OK	on success. Otherwise the related error code or MML_ERR.

The documentation for this class was generated from the following file:

- ut_class_window.h

12.8 CSurface< NUM_BUFFERS > Class Template Reference

#include <ut_class_surface.h>

Public Member Functions

- CSurface ()
- void Init ()
- ~CSurface ()
- MM_ERROR Delete ()
- virtual MM_ERROR CreateBuffer (const MM_U32 width, const MM_U32 height, const MML_GDC_SURF_FORMAT format=MML_GDC_SURF_FORMAT_R8G8B8A8, MM_U32 MaxSize=0)
- virtual MM_ERROR CreateBuffer (const MM_U32 width, const MM_U32 height, MM_U32 bit_red, MM_U32 bit_green, MM_U32 bit_blue, MM_U32 bit_alpha)
- virtual MM_ERROR CreateGrayBuffer (const MM_U32 width, const MM_U32 height, MM_U32 bit_color, MM_U32 bit_alpha)
- virtual MM_ERROR SurfLoadBitmap (const void *pImage, MM_BOOL bCopyToVRAM=MM_FALSE)
- virtual MM_ERROR Copy (MML_GDC_SURFACE surface)
- MM_S32 GetWidth ()
- MM_S32 GetHeight ()
- MM_BOOL HasBuffer ()
- MML_GDC_SURFACE GetSurface ()
- MML_GDC_SURFACE GetHandle ()
- operator MML_GDC_SURFACE ()
- MM_U32 GetBufferBufferCnt ()
- MML_GDC_SURFACE GetSurface (int id)
- void Swap ()

Protected Attributes

- MML_GDC_SURFACE_CONTAINER m_buffer [NUM_BUFFERS]
- MM_U32 m_bufferIdx
- MM_BOOL m_bHasBuffer

12.8.1 Detailed Description

template<unsigned int NUM_BUFFERS = 1>class CSurface< NUM_BUFFERS >

Class CSurface.

12.8.2 Constructor & Destructor Documentation

12.8.2.1 ~CSurface ( ) [inline]

Class CSurface destructor.

12.8.3 Member Function Documentation

12.8.3.1 MM_U32 GetBufferBufferCnt ( ) [inline]

Get number of surfaces on the class.

Return values

return	The number of surfaces in the class object.

12.8.3.2 MML_GDC_SURFACE GetHandle ( ) [inline]

Return values

return	The surface object.

12.8.3.3 MM_S32 GetHeight ( ) [inline]

Return values

return	Surface height.

12.8.3.4 MML_GDC_SURFACE GetSurface ( ) [inline]

Return values

return	The surface object.

12.8.3.5 MML_GDC_SURFACE GetSurface (int id) [inline]

Get a dedicated surface.

Parameters

in	id	0: return the current foreground buffer. If id > 0 the function will return the (id) next foreground buffer.

Return values

return	The surface object.

12.8.3.6 MM_S32 GetWidth ( ) [inline]

Get the surface width.

Return values

return	Surface width.

12.8.3.7 MM_BOOL HasBuffer ( ) [inline]

Returns whether or not the surface object owns the memory of the surfaces.

Note

- *Some member functions like CreateBuffer() allocate VRAM other function like Copy() only point to a memory buffer.*

Return values

MM_TRUE	if the surface owns the buffer otherwise MM_FALSE.

12.8.3.8 operator MML_GDC_SURFACE ( ) [inline]

Get the (foreground) surfaces.

Return values

return	The surface object.

12.8.3.9 void Swap ( ) [inline]

Toggle the foreground and background buffer.

Return values

MML_OK	on success. Otherwise the related error code.

12.8.4 Field Documentation

12.8.4.1 MM_BOOL m_bHasBuffer [protected]

MM_TRUE if the buffer was allocated in this class. In this case the destructor must free the memory.

12.8.4.2 MML_GDC_SURFACE_CONTAINER m_buffer[NUM_BUFFERS] [protected]

MML_GDC_SURFACE_CONTAINER object(s) used to describe the buffers(s).

12.8.4.3 MM_U32 m_bufferIdx [protected]

The index of the current render buffer.

The documentation for this class was generated from the following file:

- ut_class_surface.h

12.9 CSurfaceWindow< NUM_BUFFERS > Class Template Reference

#include <ut_class_window.h>

Public Member Functions

- MM_ERROR CreateBuffer (const MML_GDC_SURF_FORMAT format=MML_GDC_SURF_FORMAT_R8G8B8A8, MM_U32 MaxSize=0)
- MM_ERROR CreateBuffer (MM_U32 bit_red, MM_U32 bit_green, MM_U32 bit_blue, MM_U32 bit_alpha)
- MM_ERROR CreateGrayBuffer (MM_U32 bit_color, MM_U32 bit_alpha)
- virtual MM_ERROR Swap ()
- virtual MM_ERROR Close ()

Data Fields

- CSurface< NUM_BUFFERS > m_surface

12.9.1 Detailed Description

template<unsigned int NUM_BUFFERS>class CSurfaceWindow< NUM_BUFFERS >

The class CSurfaceWindow represents a CWindow with one or more pixel buffers. The pixel buffers can be used to store a (rendered) image that will be showed in the Window after calling Swap

12.9.2 Member Function Documentation

12.9.2.1 virtual MM_ERROR Close ( ) [inline], [virtual]

Close the Window.

Return values

MML_OK	on success. Otherwise the related error code or MML_ERR.

Reimplemented from CWindow.

12.9.2.2 MM_ERROR CreateBuffer (const MML_GDC_SURF_FORMAT format = MML_GDC_SURF_FORMAT_R8G8B8A8, MM_U32 MaxSize = 0) [inline]

Create one or more pixel buffers with the size of the window

Parameters

in	format	define the color format of the buffer
in	MaxSize	experimental: if a size != 0 is defined the function tries to create compressed buffers equal or smaller than MaxSize. (see Image Compression)

Return values

MML_OK	on success. Otherwise the related error code or MML_ERR.

12.9.2.3 MM_ERROR CreateBuffer (MM_U32 bit_red, MM_U32 bit_green, MM_U32 bit_blue, MM_U32 bit_alpha) [inline]

Create one or more pixel buffers with the size of the window.

Parameters

in	bit_red	Number of red bits in the buffer(s).
in	bit_green	Number of green bits in the buffer(s).
in	bit_blue	Number of blue bits in the buffer(s).
in	bit_alpha	Number of alpha bits in the buffer(s).

Return values

MML_OK	on success. Otherwise the related error code or MML_ERR.

12.9.2.4 MM_ERROR CreateGrayBuffer (MM_U32 bit_color, MM_U32 bit_alpha) [inline]

Create one or more pixel buffers with the size of the window and a grey pixel format.

Parameters

in	bit_color	Number of grey bits in the buffer(s).
in	bit_alpha	Number of alpha bits in the buffer(s).

Return values

MML_OK	on success. Otherwise the related error code or MML_ERR.

12.9.2.5 virtual MM_ERROR Swap ( ) [inline], [virtual]

Push the current buffer to the display and select the next buffer (if any) for next drawing operations.

Return values

MML_OK	on success. Otherwise the related error code or MML_ERR.

12.9.3 Field Documentation

12.9.3.1 CSurface<NUM_BUFFERS> m_surface

The surface (array) for this window.

The documentation for this class was generated from the following file:

- ut_class_window.h

12.10 CWindow Class Reference

#include <ut_class_window.h>

Public Member Functions

- CWindow ()
- ~CWindow ()
- virtual MM_ERROR Open (MML_GDC_DISPLAY display, MM_S32 x=0, MM_S32 y=0, MM_U32 w=0, MM_U32 h=0, MML_GDC_DISP_LAYER layerId=MML_GDC_DISP_LAYER_0, MML_GDC_DISP_SUB_LAYER sub_layerID=MML_GDC_DISP_SUB_LAYER_DEFAULT, MM_U32 features=0, MM_U32 blend_mode=0)
- virtual MM_ERROR Close ()
- virtual MM_BOOL SyncReady ()
- virtual MM_ERROR Commit ()
- virtual MM_ERROR SetSurface (MML_GDC_SURFACE surf)
- unsigned int GetWidth ()
- unsigned int GetHeight ()
- MML_GDC_DISPLAY GetDisplay ()
- MML_GDC_DISP_WINDOW GetWindowHandle ()
- operator MML_GDC_DISP_WINDOW ()
- MML_GDC_SYNC GetSync ()

Data Fields

- MML_GDC_DISP_WINDOW m_win
- MML_GDC_DISPLAY m_display
- MML_GDC_DISP_WINDOW_PROPERTIES m_windowProp

12.10.1 Detailed Description

Generic Window class (Base class for some specialized derived classes

12.10.2 Constructor & Destructor Documentation

12.10.2.1 CWindow ( ) [inline]

Class CWindow constructor.

12.10.2.2 ~CWindow ( ) [inline]

Class CWindow destructor.

12.10.3 Member Function Documentation

12.10.3.1 virtual MM_ERROR Close ( ) [inline], [virtual]

Close the window.

Return values

MML_OK	on success. Otherwise the related error code or MML_ERR.

Reimplemented in CSurfaceWindow< NUM_BUFFERS >, and CSurfaceWindow< 1 >.

12.10.3.2 virtual MM_ERROR Commit ( ) [inline], [virtual]

Apply all changes.

Return values

MML_OK	on success. Otherwise the related error code or MML_ERR.

12.10.3.3 MML_GDC_DISPLAY GetDisplay ( ) [inline]

Return values

Return	the display object of the window.

12.10.3.4 unsigned int GetHeight ( ) [inline]

Return values

Return	the height of the window.

12.10.3.5 MML_GDC_SYNC GetSync ( ) [inline]

Get the sync object of this window.

12.10.3.6 unsigned int GetWidth ( ) [inline]

Return values

Return	the width of the window.

12.10.3.7 MML_GDC_DISP_WINDOW GetWindowHandle ( ) [inline]

Return values

Return	the window object.

12.10.3.8 virtual MM_ERROR Open (MML_GDC_DISPLAY display, MM_S32 x = 0, MM_S32 y = 0, MM_U32 w = 0, MM_U32 h = 0, MML_GDC_DISP_LAYER layerId = MML_GDC_DISP_LAYER_0, MML_GDC_DISP_SUB_LAYER sub_layerID = MML_GDC_DISP_SUB_LAYER_DEFAULT, MM_U32 features = 0, MM_U32 blend_mode = 0) [inline], [virtual]

Open the window.

Parameters

in	display	A display object that will be used to open the window.
in	x	X position offset of the upper left window corner relative to the display screen.
in	y	Y position offset of the upper left window corner relative to the display screen.
in	w	Width of the window.
in	h	Height of the window.
in	layerId	Layer ID of the window.
in	sub_layerID	Sub-Layer ID of the window.
in	features	Requested features for the window see MML_GDC_DISP_WINDOW_PROPERTIES.
in	blend_mode	Starting blend mode for the window see mmlGdcDispWinSetBlendMode().

Return values

MML_OK	on success. Otherwise the related error code or MML_ERR.

12.10.3.9 operator MML_GDC_DISP_WINDOW ( ) [inline]

Return values

Return	the window object.

12.10.3.10 virtual MM_ERROR SetSurface (MML_GDC_SURFACE surf) [inline], [virtual]

Set a new surface to the window and apply changes.

Parameters

in	surf	The new surface to be shown.

Return values

MML_OK	on success. Otherwise the related error code or MML_ERR.

12.10.3.11 virtual MM_BOOL SyncReady ( ) [inline], [virtual]

Check the sync object of this window. TRUE: window is ready, FALSE window is still bussy.

12.10.4 Field Documentation

12.10.4.1 MML_GDC_DISPLAY m_display

The display object used by this class instance.

12.10.4.2 MML_GDC_DISP_WINDOW m_win

The window object used by this class instance.

12.10.4.3 MML_GDC_DISP_WINDOW_PROPERTIES m_windowProp

The MML_GDC_DISP_WINDOW_PROPERTIES structure used to create this window. The documentation for this class was generated from the following file:

- ut_class_window.h

12.11 RLAD::Frame Class Reference

#include <ut_class_rlad.h>

Data Structures

- struct Pixel

Public Member Functions

- Frame (unsigned _width, unsigned _height)
- Pixel & Read ()
- void ResetRead ()
- void Write (const Pixel &pix)
- void ResetWrite ()
- Pixel GetPixel (unsigned x, unsigned y) const

12.11.1 Detailed Description

The class Frame is used to store the uncompressed image

12.11.2 Constructor & Destructor Documentation

12.11.2.1 Frame (unsigned _width, unsigned _height) [inline]

Constructor

Parameters

in	_width	Frame width
in	_height	Frame height

12.11.3 Member Function Documentation

12.11.3.1 Pixel GetPixel (unsigned x, unsigned y) const [inline]

Get Pixel at position x, y

Parameters

in	x	X position
in	y	Y position

Return values

Pixel	

12.11.3.2 Pixel& Read ( ) [inline]

Return values

return	Read and return one pixel

12.11.3.3 void ResetRead ( ) [inline]

Reset read operation for frame start

12.11.3.4 void ResetWrite ( ) [inline]

Reset write operation for frame start

12.11.3.5 void Write (const Pixel & pix) [inline]

Write one pixel

Parameters

in	pix	Pixel

The documentation for this class was generated from the following file:

- ut_class_rlad.h

12.12 MML_GDC_DISP_MODE_LINE Struct Reference

#include <mml_gdc_display.h>

Data Fields

- MM_FLOAT pixelClock
- MM_U32 horDisplayPeriod
- MM_U32 horPulseStart
- MM_U32 horPulseEnd
- MM_U32 horTotal
- MM_U32 vertDisplayPeriod
- MM_U32 vertPulseStart
- MM_U32 vertPulseEnd
- MM_U32 vertTotal
- MM_U32 DCKDelay
- MML_GDC_DISP_DCK_INVERT_ENABLE DCKInvertEnable
- MM_U32 syncPolarity

12.12.1 Detailed Description

Data type used to specify custom timing for a display mode.

12.12.2 Field Documentation

12.12.2.1 MM_U32 DCKDelay

Number of display clock delay, default no additional delay, value is in [0, 16].

12.12.2.2 MML_GDC_DISP_DCK_INVERT_ENABLE DCKInvertEnable

Enable inversion of display clock, default set as not inverted.

12.12.2.3 MM_U32 horDisplayPeriod

Horizontal Display Period - Illuminated area.

12.12.2.4 MM_U32 horPulseEnd

Number of the dot when the sync pulse ends.

12.12.2.5 MM_U32 horPulseStart

Number of the dot when the sync pulse starts.

12.12.2.6 MM_U32 horTotal

Total horizontal.

12.12.2.7 MM_FLOAT pixelClock

Pixel clock in units of MHz.

12.12.2.8 MM_U32 syncPolarity

Bit field combination of polarity control possibilities:

MML_GDC_DISP_HSYNC_LOW / MML_GDC_DISP_HSYNC_HIGH MML_GDC_DISP_VSYNC_LOW / MML_GDC_DISP_VSYNC_HIGH MML_GDC_DISP_DE_LOW / MML_GDC_DISP_DE_HIGH MML_GDC_DISP_RGB_LOW / MML_GDC_DISP_RGB_HIGH.

Default value:

MML_GDC_DISP_HSYNC_LOW | MML_GDC_DISP_VSYNC_LOW | MML_GDC_DISP_DE_HIGH | MML_GDC_DISP_RGB_LOW.

12.12.2.9 MM_U32 vertDisplayPeriod

Vertical display period - Illuminated area.

12.12.2.10 MM_U32 vertPulseEnd

Vertical sync end position.

12.12.2.11 MM_U32 vertPulseStart

Vertical sync pulse start position.

12.12.2.12 MM_U32 vertTotal

Total vertical lines.

The documentation for this struct was generated from the following file:

- mml_gdc_display.h

12.13 MML_GDC_DISP_PROPERTIES Struct Reference

#include <mml_gdc_display.h>

Data Fields

- MML_GDC_DISP_CONTROLLER outputController
- MML_GDC_DISP_MODE displayMode
- MM_U32 xResolution
- MM_U32 yResolution
- MM_U32 refreshRate
- MM_U32 fcvm
- MML_GDC_DISP_MODE_LINE * modeLine
- MML_GDC_DISP_TCON_PROPERTIES * pDISP_TCON_PROPS
- MM_U32 countTconProps

12.13.1 Detailed Description

Data type used to configure a display controller. There are 3 options to configure the display:

Option 1: Specify one of the predefined resolutions in xResolution, yResolution, refreshRate:

- 320x240@60 Hz
- 480x272@60 Hz
- 640x480@60 Hz
- 800x480@60 Hz
- 800x600@60 Hz
- 1024x768@60 Hz
- 1280x720@60 Hz
- 1600x600@60 Hz
- 1280x800@60 Hz
- 1920x768@60 Hz
- 1280x1024@60 Hz
- 1600x900@60 Hz
- 1920x1080@60 Hz

Option 2: Specify a custom resolution in xResolution, yResolution, refreshRate and set the timing parameters in the modeLine structure.

Option 3: In addition to Option 1 or 2, provide an array of TCON register address/value pairs (refer to hardware manual for a description of the timing controller registers). TCON is only supported by display controller 0 MML_GDC_DISP_CONTROLLER_0.

12.13.2 Field Documentation

12.13.2.1 MM_U32 countTconProps

Number of TCON registers to be programmed. Must be zero if no TCON is used. Must be 0 if it is not display controller 0.

12.13.2.2 MML_GDC_DISP_MODE displayMode

Single screen, dual screen or dual view.

12.13.2.3 MM_U32 fcvm

Set to a non-zero value to have the driver use the modeLine settings specified by modeLine.

12.13.2.4 MML_GDC_DISP_MODE_LINE* modeLine

Custom display timing information.

12.13.2.5 MML_GDC_DISP_CONTROLLER outputController

Must be MML_GDC_DISP_CONTROLLER_0.

12.13.2.6 MML_GDC_DISP_TCON_PROPERTIES* pDISP_TCON_PROPS

Pointer to TCON register/value structure array. Must be NULL if no TCON is used. Must be NULL if it is not display controller 0.

12.13.2.7 MM_U32 refreshRate

Refresh rate in Hz (60, 75, 85, etc.).

12.13.2.8 MM_U32 xResolution

Horizontal resolution (640, 800, 1024, etc.).

12.13.2.9 MM_U32 yResolution

Vertical resolution (480, 600, 768, etc.).

The documentation for this struct was generated from the following file:

- mml_gdc_display.h

12.14 MML_GDC_DISP_TCON_PROPERTIES Struct Reference

#include <mml_gdc_display.h>

Data Fields

- MM_U32 address
- MM_U32 value

12.14.1 Detailed Description

Data type used to program timing controller (TCON) registers.

12.14.2 Field Documentation

12.14.2.1 MM_U32 address

Address of the TCON register.

12.14.2.2 MM_U32 value

Value of the TCON register.

The documentation for this struct was generated from the following file:

- mml_gdc_display.h

12.15 MML_GDC_DISP_WINDOW_PROPERTIES Struct Reference

#include <mml_gdc_display.h>

Data Fields

- MML_GDC_DISP_OUTPUT_SCREEN outputScreen
- MM_U32 topLeftX
- MM_U32 topLeftY
- MM_U32 width

- MM_U32 height
- MML_GDC_DISP_LAYER layerId
- MM_U32 features
- MML_GDC_DISP_SUB_LAYER sub_layerId

12.15.1 Detailed Description

Data type used to specify window creation parameters.

12.15.2 Field Documentation

12.15.2.1 MM_U32 features

Features requested by the layer, the related parameter can be a bit field combination of:

- MML_GDC_DISP_FEATURE_INDEX_COLOR.
- MML_GDC_DISP_FEATURE_DECODE.
- MML_GDC_DISP_FEATURE_MULTI_LAYER.

12.15.2.2 MM_U32 height

Height of the window.

12.15.2.3 MML_GDC_DISP_LAYER layerId

Layer to use for the window (see MML_GDC_DISP_LAYER).

12.15.2.4 MML_GDC_DISP_OUTPUT_SCREEN outputScreen

Which output screen should the window be created on.

12.15.2.5 MML_GDC_DISP_SUB_LAYER sub_layerId

Sub-Layer to use for for windows with feature MML_GDC_DISP_FEATURE_MULTI_LAYER.

12.15.2.6 MM_U32 topLeftX

Top left X coordinate of the window on the display.

12.15.2.7 MM_U32 topLeftY

Top left Y coordinate of the window on the display.

12.15.2.8 MM_U32 width

Width of the window.

The documentation for this struct was generated from the following file:

- mml_gdc_display.h

12.16 MML_GDC_PE_CONTEXT_CONTAINER Struct Reference

#include <mml_gdc_pixeng.h>

Data Fields

- MM_U32 reserved [84]

12.16.1 Detailed Description

The pixel engine context container.

12.16.2 Field Documentation

12.16.2.1 MM_U32 reserved[84]

Reserved memory needed for any context container.

The documentation for this struct was generated from the following file:

- mml_gdc_pixeng.h

12.17 MML_GDC_SURFACE_CONTAINER Struct Reference

#include <mml_gdc_surfman.h>

Data Fields

- MM_U32 reserved [15]

12.17.1 Detailed Description

The surface object container

12.17.2 Field Documentation

12.17.2.1 MM_U32 reserved[15]

Reserved memory needed for any surface container

The documentation for this struct was generated from the following file:

- mml_gdc_surfman.h

12.18 MML_GDC_SYNC_CONTAINER Struct Reference

#include <mml_gdc_sync.h>

Data Fields

- MM_U32 reserved [3]

12.18.1 Detailed Description

Data type to refer to a sync object.

12.18.2 Field Documentation

12.18.2.1 MM_U32 reserved[3]

Reserved memory needed for any sync container.

The documentation for this struct was generated from the following file:

- mml_gdc_sync.h

12.19 MML_GDC_SYSINIT_INFO Struct Reference

#include <mml_gdc_sysinit.h>

Data Fields

- MM_U32 ResourceLock
- MM_U32 GfxPll

12.19.1 Detailed Description

Data type used to program timing controller (TCON) registers

12.19.2 Field Documentation

12.19.2.1 MM_U32 GfxPll

Frequency of GFX PLL (for Pixel Clock generation) in Hertz, Default=200000000

12.19.2.2 MM_U32 ResourceLock

Bitfield that describes resources allocated by safety driver

The documentation for this struct was generated from the following file:

- mml_gdc_sysinit.h

12.20 RLAD::Package Struct Reference

#include <ut_class_rlad.h>

Public Member Functions

- Package (RLAD ∗_cfg)
- void Reset ()
- void Add (const RLAD::Frame::Pixel &pix)
- void Serialize (queue< RLAD::Frame::Pixel > &fifo, RLAD::BitStream &bs, unsigned &pkg, unsigned &x, unsigned &y)

Data Fields

- RLAD ∗ cfg
- bool delta
- unsigned pcnt
- unsigned cbpc [NUM_C]
- unsigned cbpp

- unsigned size
- unsigned cofs [NUM_C]
- unsigned crange [NUM_C]
- unsigned start [NUM_C]
- unsigned prev [NUM_C]
- int dmin [NUM_C]
- int dmax [NUM_C]

12.20.1 Detailed Description

Helper structure for RLA compression

12.20.2 Field Documentation

12.20.2.1 unsigned cbpc[NUM_C]

bits per compressed component

12.20.2.2 unsigned cbpp

bits per compressed pixels

12.20.2.3 RLAD* cfg

Reference

12.20.2.4 unsigned cofs[NUM_C]

offset package

12.20.2.5 bool delta

package type

12.20.2.6 unsigned pcnt

pixel count

12.20.2.7 unsigned size

package size in bits

12.20.2.8 unsigned start[NUM_C]

delta package

The documentation for this struct was generated from the following file:

- ut_class_rlad.h

12.21 RLAD::Frame::Pixel Struct Reference

#include <ut_class_rlad.h>

Data Fields

- unsigned col [NUM_C]

12.21.1 Detailed Description

Helper structure to store one pixel

12.21.2 Field Documentation

12.21.2.1 unsigned col[NUM_C]

array with bit size for all components

The documentation for this struct was generated from the following file:

- ut_class_rlad.h

12.22 RLAD Class Reference

#include <ut_class_rlad.h>

Data Structures

- class BitStream
- class Frame
- struct Package

Public Types

- enum { NUM_C = 4 }
- enum { MAX_BPC = 8 }
- enum { CNT_RLAD = 8 }
- enum { MAX_CNT_RLA = 32 }
- enum Mode {MODE_RLAD, MODE_RLAD_UNIFORM, MODE_RLA, MODE_RL,NUM_MODE }

Public Member Functions

- unsigned cbpc_width (unsigned i) const
- unsigned cnt_width () const
- unsigned cwrap (unsigned i) const
- unsigned max_code (unsigned i) const
- unsigned header_size () const
- unsigned buffer_size () const
- unsigned bpp () const
- unsigned cbpp_max () const
- unsigned image_size () const
- double compression_rate () const
- bool Encode (Frame &f, BitStream &bs)
- bool Decode (BitStream &bs, Frame &f)

Data Fields

- enum RLAD::Mode mode
- unsigned width
- unsigned height
- unsigned bpc [NUM_C]
- unsigned cbpc_max [NUM_C]
- bool decode_BufferTooSmall
- bool decode_BufferTooLarge

Protected Member Functions

- bool Encode_Lossy (Frame &f, BitStream &bs)
- bool Encode_Lossless (Frame &f, BitStream &bs)
- void set_pbpc (unsigned *pbpc, unsigned *cbpc, unsigned &credit_cnt, unsigned pcnt)

Static Protected Member Functions

- static unsigned SpatialDither (unsigned data_in, unsigned size_in, unsigned size_out, unsigned x, unsigned y, bool exact)
- static unsigned MSBitReplication (unsigned data_in, unsigned size_in, unsigned size_out)
- static int ClampToBpc (unsigned int data_in, unsigned bpc)
- static unsigned Log2 (unsigned t)

12.22.1 Detailed Description

This class contains sample code for compression

12.22.2 Member Enumeration Documentation

12.22.2.1 enum Mode

configuration

Enumerator

MODE_RLAD_UNIFORM Proprietary (lossy with upper limit for compression rate) Proprietary (lossy with fixed compression rate)

MODE_RLA Proprietary (lossless)

MODE_RL Standard RL format according to TGA spec (for backward compatibility)

12.22.3 Member Function Documentation

12.22.3.1 unsigned bpp ( ) const [inline]

return sum of component bpp

12.22.3.2 unsigned buffer_size ( ) const

calc buffer size

12.22.3.3 unsigned cbpc_width (unsigned i) const [inline]

bit width of cbpc fields in package headers

12.22.3.4 unsigned cbpp_max ( ) const [inline]

return sum of compressed component bpp

12.22.3.5 unsigned cnt_width ( ) const [inline]

return max bit size

12.22.3.6 double compression_rate ( ) const [inline]

Return compression rate

12.22.3.7 unsigned cwrap (unsigned i) const [inline]

return component size

12.22.3.8 bool Decode (BitStream & bs, Frame & f)

Decode image

Parameters

in	bs	BitStream class containing the compressed data
out	f	Store the uncompressed image

Return values

True	if successful otherwise false

12.22.3.9 bool Encode (Frame & f, BitStream & bs)

Encode image

Parameters

in	f	Uncompressed image
out	bs	BitStream class storing the compressed data

Return values

True	if successful otherwise false

12.22.3.10 unsigned header_size ( ) const

calc header size

12.22.3.11 unsigned image_size ( ) const [inline]

return uncompressed image size

12.22.3.12 unsigned max_code (unsigned i) const [inline]

return max component value

12.22.4 Field Documentation

12.22.4.1 unsigned bpc[NUM_C]

bits per channel and pixel of uncompressed image

12.22.4.2 unsigned cbpc_max[NUM_C]

max value for compressed bits per channel and pixel (RLAD only)

12.22.4.3 bool decode_BufferTooLarge

Buffer larger than required

12.22.4.4 bool decode_BufferTooSmall

Buffer too small for decompression

12.22.4.5 unsigned height

frame dimension height in pixels

12.22.4.6 enum RLAD::Mode mode

store the compression mode

12.22.4.7 unsigned width

frame dimension width in pixels

The documentation for this class was generated from the following file:

- ut_class_rlad.h

13. File Documentation

13.1 flash_resource.h File Reference

Include this file before the definition of a bitmap.

13.1.1.1 Detailed Description

Include this file before the definition of a bitmap.

13.2 mm_defines.h File Reference

Common macro definitions for all modules.

#include "mm_types.h"

Macros

- #define MM_ERRCODE(err) ((MM_ERROR)(err))
- #define MM_MODULEID(moduleId) ((MM_MODULE)(moduleId))
- #define MML_ERR MM_ERRCODE(0x3FFFFFFF)
- #define MMD_ERR MM_ERRCODE(0x7FFFFFFF)
- #define MML_OK MM_ERRCODE(0x0)
- #define MMD_OK MM_ERRCODE(0x0)
- #define MM_FALSE ((MM_BOOL) 0)
- #define MM_TRUE ((MM_BOOL) 1)
- #define NULL ((void ∗)0)
- #define MM_BIT(x) (1U<<(x))
- #define MM_PTR_TO_ADDR(x) (MM_ADDR)(x)
- #define MM_ADDR_TO_PTR(x) (void∗)(x)
- #define MM_ADDR_TO_UINT32(x) (MM_U32)(x)
- #define MM_UINT32_TO_ADDR(x) (MM_ADDR)(x)
- #define MM_PTR_TO_UINT32(x) (MM_U32)(x)
- #define MM_UINT32_TO_PTR(x) (void∗)(x)
- #define MM_ADDR_TO_UINT32PTR(x) (MM_U32∗)(x)
- #define MM_ADDR_TO_SINT32PTR(x) (MM_S32∗)(x)
- #define MM_IO_IRIS_SUBSYSTEM 0xD0A00000U
- #define MM_IO_IRIS_CORE 0xD0A10000U
- #define NULL_FUNCTION ((void) 0)
- #define UNUSED_PARAMETER(x) (void)(x)

13.2.1 Detailed Description

Common macro definitions for all modules.

13.3 mm_gdc_erp.h File Reference

Error Reporting API.

Enumerations

- enum MM_ERP_MESSAGE_LEVEL { MM_ERP_LEVEL_NOTHING = 0U, MM_ERP_LEVEL_ERROR, MM_ERP_LEVEL_WARNING, MM_ERP_LEVEL_INFO }
- enum MM_ERP_MESSAGE_CHANNEL_PROP { MM_ERP_CH_OFF = 0U, MM_ERP_CH_ON }
- enum MM_ERP_MESSAGE_DEST { MM_ERP_CH_STDOUT = 0U, MM_ERP_CH_BUFFER }

13.3.1 Detailed Description

Error Reporting API.

13.4 mm_gdc_errors.h File Reference

Error Codes for the Basic Graphics modules.

#include "mm_defines.h"

Macros

Error codes for Config API

- #define MML_ERR_GDC_CONFIG_INVALID_PARAMETER MM_ERRCODE(0x21008001)
- #define MML_ERR_GDC_CONFIG_INTERNAL_ERROR MM_ERRCODE(0x21008002)
- #define MML_ERR_GDC_CONFIG_INVALID_ADDRESS MM_ERRCODE(0x21008003)

Error codes for Display API

- #define MML_ERR_GDC_DISP_DEVICE_NOT_FOUND MM_ERRCODE(0x21001001)
- #define MML_ERR_GDC_DISP_DISPLAY_ALREADY_OPEN MM_ERRCODE(0x21001002)
- #define MML_ERR_GDC_DISP_INVALID_ARG MM_ERRCODE(0x21001003)
- #define MML_ERR_GDC_DISP_UNSUPPORTED_MODE MM_ERRCODE(0x21001004)
- #define MML_ERR_GDC_DISP_DEVICE_INIT_FAILED MM_ERRCODE(0x21001005)
- #define MML_ERR_GDC_DISP_DEVICE_CLOSE_FAILED MM_ERRCODE(0x21001006)
- #define MML_ERR_GDC_DISP_OUT_OF_SYSTEM_MEMORY MM_ERRCODE(0x21001007)
- #define MML_ERR_GDC_DISP_LAYER_ALREADY_USED MM_ERRCODE(0x21001008)
- #define MML_ERR_GDC_DISP_WRONG_PIXEL_FORMAT MM_ERRCODE(0x21001009)
- #define MML_ERR_GDC_DISP_WRONG_STRIDE MM_ERRCODE(0x21001011)
- #define MML_ERR_GDC_DISP_WRONG_WINDOW MM_ERRCODE(0x21001012)
- #define MML_ERR_GDC_DISP_WRONG_INDEX_WINDOW MM_ERRCODE(0x21001013)
- #define MML_ERR_GDC_DISP_FAILED MM_ERRCODE(0x21001014)
- #define MML_ERR_GDC_DISP_WRONG_YC_WINDOW MM_ERRCODE(0x21001015)
- #define MML_ERR_GDC_DISP_WRONG_TCON_PARAMS MM_ERRCODE(0x21001016)
- #define MML_ERR_GDC_DISP_DISPLAY_MODE_MISSMATCH MM_ERRCODE(0x21001017)
- #define MML_ERR_GDC_DISP_INVALID_SCALING MM_ERRCODE(0x21001018)
- #define MML_ERR_GDC_DISP_INVALID_BLENDING MM_ERRCODE(0x21001019)
- #define MML_ERR_GDC_DISP_INVALID_CLUTDATA MM_ERRCODE(0x2100101a)
- #define MML_ERR_GDC_DISP_INVALID_DIMENSION MM_ERRCODE(0x2100101c)
- #define MML_ERR_GDC_DISP_DEV_BUSY MM_ERRCODE(0x21001020)

Error codes for Error Reporting API

- #define MML_ERR_ERP_ALREADY_INITIALIZED MM_ERRCODE(0x2100F000)
- #define MML_ERR_ERP_NOT_INITIALIZED MM_ERRCODE(0x2100F001)
- #define MML_ERR_ERP_INVALID_PARAMETER MM_ERRCODE(0x2100F002)

Error codes for Pixel Engine API

- #define MML_ERR_GDC_PE_OUT_OF_SPACE MM_ERRCODE(0x2100D001)
- #define MML_ERR_GDC_PE_INVALID_CONTEXT MM_ERRCODE(0x2100D002)
- #define MML_ERR_GDC_PE_INVALID_TARGET MM_ERRCODE(0x2100D003)
- #define MML_ERR_GDC_PE_INVALID_SURFACE_OBJECT MM_ERRCODE(0x2100D004)
- #define MML_ERR_GDC_PE_INVALID_ADDRESS MM_ERRCODE(0x2100D005)
- #define MML_ERR_GDC_PE_INVALID_MATRIX MM_ERRCODE(0x2100D006)
- #define MML_ERR_GDC_PE_INVALID_DIMENSION MM_ERRCODE(0x2100D007)
- #define MML_ERR_GDC_PE_INVALID_STRIDE MM_ERRCODE(0x2100D008)
- #define MML_ERR_GDC_PE_INVALID_BITS_PER_PIXEL MM_ERRCODE(0x2100D009)
- #define MML_ERR_GDC_PE_INVALID_COMPRESSION MM_ERRCODE(0x2100D010)
- #define MML_ERR_GDC_PE_INVALID_RLD_REQUEST MM_ERRCODE(0x2100D011)
- #define MML_ERR_GDC_PE_INVALID_ROP_MODE MM_ERRCODE(0x2100D012)
- #define MML_ERR_GDC_PE_INVALID_SURFACE_PARAM MM_ERRCODE(0x2100D013)
- #define MML_ERR_GDC_PE_INVALID_NO_ACTIVE_AREA MM_ERRCODE(0x2100D014)
- #define MML_ERR_GDC_PE_INVALID_ATTRIBUTE MM_ERRCODE(0x2100D015)
- #define MML_ERR_GDC_PE_INVALID_PARAMETER MM_ERRCODE(0x2100D016)
- #define MML_ERR_GDC_PE_INVALID_OPERATION MM_ERRCODE(0x2100D017)
- #define MML_ERR_GDC_PE_INVALID_MASK_PARAM MM_ERRCODE(0x2100D018)
- #define MML_ERR_GDC_PE_INVALID_SCALING MM_ERRCODE(0x2100D019)
- #define MML_ERR_GDC_PE_INVALID_STORE_COMRESSION MM_ERRCODE(0x2100D020)
- #define MML_ERR_GDC_PE_INVALID_STORE_CLUT MM_ERRCODE(0x2100D021)
- #define MML_ERR_GDC_PE_INVALID_FLOAT MM_ERRCODE(0x2100D023)
- #define MML_ERR_GDC_PE_INVALID_CLUT_OPERATION MM_ERRCODE(0x2100D024)
- #define MML_ERR_GDC_PE_INVALID_YUV_PARAM MM_ERRCODE(0x2100D028)
- #define MML_ERR_GDC_PE_INVALID_COMPRESSION_OPERATION MM_ERRCODE(0x2100D029)

Error codes for Surface Manager API

- #define MML_ERR_GDC_SURF_OUT_OF_SPACE MM_ERRCODE(0x21000001)
- #define MML_ERR_GDC_SURF_OUT_OF_VRAM MM_ERRCODE(0x21000002)
- #define MML_ERR_GDC_SURF_INVALID_SURFACE MM_ERRCODE(0x21000003)
- #define MML_ERR_GDC_SURF_INVALID_FORMAT MM_ERRCODE(0x21000004)
- #define MML_ERR_GDC_SURF_INVALID_FOR_BUFFER_OWNED MM_ERRCODE(0x21000005)
- #define MML_ERR_GDC_SURF_INVALID_ATTRIBUTE MM_ERRCODE(0x21000006)
- #define MML_ERR_GDC_SURF_ERROR_ADDRESS_TRANSLATION MM_ERRCODE(0x21000007)
- #define MML_ERR_GDC_SURF_INVALID_PARAMETER MM_ERRCODE(0x21000008)
- #define MML_ERR_GDC_SURF_INVALID_ADDRESS_ALIGNMENT MM_ERRCODE(0x21000009)

Error codes for Synchronization API

- #define MML_ERR_GDC_SYNC_INVALID_PARAMETER MM_ERRCODE(0x21005001)
- #define MML_ERR_GDC_SYNC_OUT_OF_MEMORY MM_ERRCODE(0x21005002)
- #define MML_ERR_GDC_SYNC_TIMEOUT MM_ERRCODE(0x21005003)
- #define MML_ERR_GDC_SYNC_INVALID MM_ERRCODE(0x21005004)

Error codes for Driver Initialization API

- #define MML_ERR_GDC_SYS_DEVICE_INIT_FAILED MM_ERRCODE(0x21009001)
- #define MML_ERR_GDC_SYS_DEVICE_CLOSE_FAILED MM_ERRCODE(0x21009002)
- #define MML_ERR_GDC_SYS_DEVICE_ALREADY_INITIALIZED MM_ERRCODE(0x21009003)
- #define MML_ERR_GDC_SYS_DEVICE_NOT_YET_INITIALIZED MM_ERRCODE(0x21009004)
- #define MML_ERR_GDC_SYS_DEVICE_INVALID_PARAMETER MM_ERRCODE(0x21009005)
- #define MML_ERR_GDC_SYS_DEVICE_WRONG_ID MM_ERRCODE(0x21009006)

Error codes for Writeback API

- #define MML_ERR_GDC_WB_DEVICE_BUSY MM_ERRCODE(0x21004001)
- #define MML_ERR_GDC_WB_INVALID_PARAMETER MM_ERRCODE(0x21004002)

Error codes for Internal function calls

- #define MML_ERR_GDC_CARD_DEV_NOT_ENABLED MM_ERRCODE(0x21007001)
- #define MML_ERR_GDC_CARD_DEV_ENABLED MM_ERRCODE(0x21007002)
- #define MML_ERR_GDC_CARD_DEV_NOTSUPPORTED MM_ERRCODE(0x21007003)
- #define MML_ERR_GDC_CARD_ACCESS_FAILED MM_ERRCODE(0x21007004)
- #define MML_ERR_GDC_CARD_THREAD_LIMIT MM_ERRCODE(0x21007005)
- #define MML_ERR_GDC_CARD_TIMEOUT_EXPIRED MM_ERRCODE(0x21007006)
- #define MML_ERR_GDC_CARD_DEV_BUSY MM_ERRCODE(0x21007007)
- #define MML_ERR_GDC_IRIS_CMD_SEQ_ARG_ERROR MM_ERRCODE(0x2100B001)
- #define MML_ERR_GDC_IRIS_CMD_SEQ_INVALID_ADDRESS MM_ERRCODE(0x2100B002)
- #define MML_ERR_GDC_IRIS_CMD_SEQ_INVALID_BUFFER_SIZE MM_ERRCODE(0x2100B003)
- #define MML_ERR_GDC_IRIS_CMD_SEQ_FIFO_UNINITIALIZED MM_ERRCODE(0x2100B004)
- #define MML_ERR_GDC_IRIS_CMD_SEQ_COMMAND_QUEUE_FULL MM_ERRCODE(0x2100B005)
- #define MMD_ERR_GDC_DISP_ARG_ERROR MM_ERRCODE(0x11001003)
- #define MML_ERR_GDC_INT_OUT_OF_RANGE MM_ERRCODE(0x21010001)
- #define MMD_ERR_GDC_INT_OUT_OF_RANGE MM_ERRCODE(0x11010001)
- #define MML_ERR_GDC_IRIS_MATH_INVALID_FLOAT MM_ERRCODE(0x21003001)
- #define MML_ERR_GDC_IRIS_MATH_INVALID_MATRIX MM_ERRCODE(0x21003002)
- #define MML_ERR_RES_UNKNOWN MM_ERRCODE(0x2100A000)
- #define MML_ERR_RES_EXCEEDED_MAXIMUM_USAGE MM_ERRCODE(0x2100A001)
- #define MML_ERR_RES_USAGE_COUNT_ZERO MM_ERRCODE(0x2100A002)
- #define MML_ERR_RES_MAN_ALREADY_INITIALIZED MM_ERRCODE(0x2100A003)
- #define MML_ERR_RES_MAN_NOT_INITIALIZED MM_ERRCODE(0x2100A004)
- #define MMD_ERR_GDC_SYNC_INVALID_PARAMETER MM_ERRCODE(0x11005001)
- #define MMD_ERR_GDC_SYNC_ACCESS_FAILED MM_ERRCODE(0x11005002)
- #define MMD_ERR_GDC_SYNC_TIMEOUT MM_ERRCODE(0x11005003)

- #define MMD_ERR_GDC_CARD_DEV_BUSY MM_ERRCODE(0x11007001)
- #define MMD_ERR_GDC_CARD_TIMEOUT_EXPIRED MM_ERRCODE(0x11007002)
- #define MMD_ERR_GDC_CARD_ACCESS_FAILED MM_ERRCODE(0x11007003)
- #define MMD_ERR_GDC_CARD_TIME_INTERVAL MM_ERRCODE(0x11007004)
- #define MMD_ERR_GDC_CARD_DEV_NOTSUPPORTED MM_ERRCODE(0x11007005)

13.4.1 Detailed Description

Error Codes for the Basic Graphics modules.

13.5 mm_gdc_module_id.h File Reference

Basic Graphics module ids (common)

#include "mm_defines.h"

Macros

Module Id's

(The error reporting level can be set per module id)

Note:

kernel modules are covered by the corresponding user module

- *#define MM_ERP_MODULE_ID_GDC_ALL_USER MM_MODULEID(0x2100FFFFU)*
- *#define MM_ERP_MODULE_ID_GDC_SURFMAN_USER MM_MODULEID(0x21000000U)*
- *#define MM_ERP_MODULE_ID_GDC_DISP_USER MM_MODULEID(0x21001000U)*
- *#define MM_ERP_MODULE_ID_GDC_IRIS_USER MM_MODULEID(0x21003000U)*
- *#define MM_ERP_MODULE_ID_GDC_SYNC_USER MM_MODULEID(0x21005000U)*
- *#define MM_ERP_MODULE_ID_GDC_CARD_USER MM_MODULEID(0x21006000U)*
- *#define MM_ERP_MODULE_ID_GDC_CONFIG_USER MM_MODULEID(0x21007000U)*
- *#define MM_ERP_MODULE_ID_GDC_SYSINIT_USER MM_MODULEID(0x21008000U)*
- *#define MM_ERP_MODULE_ID_GDC_CMDSEQ_USER MM_MODULEID(0x21009000U)*
- *#define MM_ERP_MODULE_ID_GDC_PIXENG_USER MM_MODULEID(0x2100B000U)*
- *#define MM_ERP_MODULE_ID_GDC_ERP_USER MM_MODULEID(0x2100D000U)*
- *#define MM_ERP_MODULE_ID_GDC_SERVICE_USER MM_MODULEID(0x2100E000U)*

13.5.1 Detailed Description

Basic Graphics module ids (common)

13.6 mm_gdc_version.h File Reference

Basic Graphics Driver Version Numbers.

#include "mm_gdc_build_version.h"

Macros

- #define MM_GDC_MAJOR_VERSION 1U
- #define MM_GDC_MINOR_VERSION 0U

13.6.1 Detailed Description

Basic Graphics Driver Version Numbers.

13.7 mm_types.h File Reference

Basic type definitions.

Typedefs

- typedef unsigned char MM_U08
- typedef signed char MM_S08
- typedef unsigned short MM_U16
- typedef signed short MM_S16
- typedef unsigned int MM_U32
- typedef signed int MM_S32
- typedef unsigned long long MM_U64
- typedef signed long long MM_S64
- typedef char MM_CHAR
- typedef float MM_FLOAT
- typedef double MM_DOUBLE
- typedef int MM_BOOL
- typedef unsigned int MM_ADDR
- typedef MM_S32 MM_ERROR
- typedef MM_S32 MM_MODULE

13.7.1 Detailed Description

Basic type definitions.

13.8 mmd_gdc_interrupthandler.h File Reference

2D Core Interrupt Controller API

#include "mm_types.h"

Macros

- #define MM_GDC_IRIS_INT_STORE9_FRAMECOMPLETE_IRQ_CP 1U
- #define MM_GDC_IRIS_INT_EXTDST0_FRAMECOMPLETE_IRQ_CP 4U
- #define MM_GDC_IRIS_INT_DISENGCFG_FRAMECOMPLETE0_IRQ_CP 10U
- #define MM_GDC_IRIS_INT_CMDSEQ_ERROR_IRQ_CP 20U
- #define MM_GDC_IRIS_INT_FRAMEGEN0_SECSYNC_ON_IRQ_CP 27U
- #define MM_GDC_IRIS_INT_FRAMEGEN0_SECSYNC_OFF_IRQ_CP 28U

Interrupt signal irqs

These can be used in mmdGdcInterruptRegisterHandler

- #define MM_GDC_IRIS_STORE9_FRAMECOMPLETE_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_STORE9_FRAMECOMPLETE_IRQ_CP)
- #define MM_GDC_IRIS_EXTDST0_FRAMECOMPLETE_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_EXTDST0_FRAMECOMPLETE_IRQ_CP)

- #define MM_GDC_IRIS_DISENGCFG_FRAMECOMPLETE0_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_DISENGCFG_FRAMECOMPLETE0_IRQ_CP)
- #define MM_GDC_IRIS_CMDSEQ_ERROR_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_CMDSEQ_ERROR_IRQ_CP)
- #define MM_GDC_IRIS_FRAMEGEN0_SECSYNC_ON_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_FRAMEGEN0_SECSYNC_ON_IRQ_CP)
- #define MM_GDC_IRIS_FRAMEGEN0_SECSYNC_OFF_IRQ ((MM_U64)1 << MM_GDC_IRIS_INT_FRAMEGEN0_SECSYNC_OFF_IRQ_CP)

Functions

Interrupt Operations Functions

- void mmdGdcInterruptHandler (void)
  Interrupt Handler Function.
- MM_ERROR mmdGdcInterruptRegisterHandler (MM_U64 irq, void(*pHandler)(MM_U64 intrrpt))
  Set an application defined interrupt handler function.

13.8.1 Detailed Description

2D Core Interrupt Controller API

13.9 mml_gdc_config.h File Reference

Controls global graphics driver and hardware configurations.

```
#include   "mm_gdc_errors.h"
```

Enumerations

- enum MML_GDC_CONFIG_ATTR {
  MML_GDC_CONFIG_ATTR_MAJOR_VERSION = 0,
  MML_GDC_CONFIG_ATTR_MINOR_VERSION,
  MML_GDC_CONFIG_ATTR_BUILD_VERSION,
  MML_GDC_CONFIG_ATTR_MIN_INSTRUCTION_BUFFER,
  MML_GDC_CONFIG_ATTR_CURRENT_INSTRUCTION_BUFFER,
  MML_GDC_CONFIG_ATTR_DISPLAY_NOBLOCK,
  MML_GDC_CONFIG_ATTR_BUILD_TYPE
  }

Functions

- MM_ERROR mmlGdcConfigSetAttribute (MML_GDC_CONFIG_ATTR pname, MM_U32 param)
- MM_ERROR mmlGdcConfigGetAttribute (MML_GDC_CONFIG_ATTR pname, MM_U32 *pParam)

13.9.1 Detailed Description

Controls global graphics driver and hardware configurations.

13.10 mml_gdc_display.h File Reference

Display API.

#include "mm_types.h"

#include "mml_gdc_surfman.h"

#include "mml_gdc_sync.h"

#include "mm_gdc_errors.h"

Data Structures

- struct MML_GDC_DISP_MODE_LINE
- struct MML_GDC_DISP_TCON_PROPERTIES
- struct MML_GDC_DISP_PROPERTIES
- struct MML_GDC_DISP_WINDOW_PROPERTIES

Macros

Layer feature request

- #define MML_GDC_DISP_FEATURE_INDEX_COLOR (1 << 0)
- #define MML_GDC_DISP_FEATURE_DECODE (1 << 1)
- #define MML_GDC_DISP_FEATURE_MULTI_LAYER (1 << 7)

Buffer target

- #define MML_GDC_DISP_BUFF_TARGET_COLOR_BUFF (1 << 1)

Blend modes

- #define MML_GDC_DISP_BLEND_NONE (0)
- #define MML_GDC_DISP_BLEND_TRANSPARENCY (1U << 0)
- #define MML_GDC_DISP_BLEND_GLOBAL_ALPHA (1U << 1)
- #define MML_GDC_DISP_BLEND_SOURCE_ALPHA (1U << 2)
- #define MML_GDC_DISP_BLEND_SOURCE_MULTIPLY_ALPHA (1U << 4)

Polarity control.

- #define MML_GDC_DISP_HSYNC_LOW (0)
- #define MML_GDC_DISP_HSYNC_HIGH (1U << 0)
- #define MML_GDC_DISP_VSYNC_LOW (0)
- #define MML_GDC_DISP_VSYNC_HIGH (1U << 1)
- #define MML_GDC_DISP_DE_LOW (0)
- #define MML_GDC_DISP_DE_HIGH (1U << 2)
- #define MML_GDC_DISP_RGB_LOW (0)
- #define MML_GDC_DISP_RGB_HIGH (1U << 3)

Default initializer

- #define MML_GDC_DISP_PROPERTIES_INITIALIZER
- #define MML_GDC_DISP_WINDOW_PROPERTIES_INITIALIZER

Typedefs

- typedef struct MML_GDC_DISPLAY * MML_GDC_DISPLAY
- typedef struct MML_GDC_DISP_WINDOW * MML_GDC_DISP_WINDOW

Enumerations

- enum MML_GDC_DISP_CONTROLLER { MML_GDC_DISP_CONTROLLER_0 = 0 }
- enum MML_GDC_DISP_MODE {
  MML_GDC_DISP_SINGLE_SCREEN = 0,
  MML_GDC_DISP_DUAL_SCREEN,
  MML_GDC_DISP_DUAL_VIEW
  }
- enum MML_GDC_DISP_OUTPUT_SCREEN {
  MML_GDC_DISP_OUTPUT_SCREEN_PRIMARY = 0,
  MML_GDC_DISP_OUTPUT_SCREEN_SECONDARY,
  MML_GDC_DISP_OUTPUT_SCREEN_BOTH
  }
- enum MML_GDC_DISP_FILTER {
  MML_GDC_DISP_FILTER_NEAREST = 0,
  MML_GDC_DISP_FILTER_BILINEAR
  }
- enum MML_GDC_DISP_TILE_MODE {
  MML_GDC_DISP_TILE_MODE_ZERO = 0,
  MML_GDC_DISP_TILE_MODE_CONST = 1,
  MML_GDC_DISP_TILE_MODE_PAD = 2,
  MML_GDC_DISP_TILE_MODE_CLIP = 3
  }
- enum MML_GDC_DISP_LAYER { MML_GDC_DISP_LAYER_0 = 0, MML_GDC_DISP_LAYER_1 }
- enum MML_GDC_DISP_SUB_LAYER {
  MML_GDC_DISP_SUB_LAYER_DEFAULT = 0,
  MML_GDC_DISP_SUB_LAYER_1,
  MML_GDC_DISP_SUB_LAYER_2,
  MML_GDC_DISP_SUB_LAYER_3,
  MML_GDC_DISP_SUB_LAYER_4,
  MML_GDC_DISP_SUB_LAYER_5,
  MML_GDC_DISP_SUB_LAYER_6,
  MML_GDC_DISP_SUB_LAYER_7,
  MML_GDC_DISP_SUB_LAYER_8
  }
- enum MML_GDC_DISP_DCK_DELAY_ENABLE {
  MML_GDC_DISP_DCK_DELAY_OFF = 0,
  MML_GDC_DISP_DCK_DELAY_ON
  }
- enum MML_GDC_DISP_DCK_INVERT_ENABLE {
  MML_GDC_DISP_DCK_INVERT_OFF = 0,
  MML_GDC_DISP_DCK_INVERT_ON
  }
- enum MML_GDC_DISP_DITHER_ENABLE {
  MML_GDC_DISP_DITHOFF = 0,
  MML_GDC_DISP_DITHON
  }
- enum MML_GDC_DISP_DITHER_MODE {
  MML_GDC_DISP_TEMPDITH = 0,
  MML_GDC_DISP_SPATDITH = (1 << 4)
  }
- enum MML_GDC_DISP_DITHER_RANGE { MML_GDC_DISP_DITHRS11LOW = 0 }
- enum MML_GDC_DISP_DITHER_FORMAT {
  MML_GDC_DISP_DITHER108 = 0x08080800,

```
        MML_GDC_DISP_DITHER107 = 0x07070700,
        MML_GDC_DISP_DITHER106 = 0x06060600,
        MML_GDC_DISP_DITHER105 = 0x05060500
    }
```

- enum MML_GDC_DISP_CLUT_FORMAT { MML_GDC_DISP_CLUT_FORMAT_33 = 33 }
- enum MML_GDC_DISP_CMATRIX_FORMAT {
  MML_GDC_DISP_CMATRIX_FORMAT_4X3 = 0,
  MML_GDC_DISP_CMATRIX_FORMAT_5X4
  }
- enum MML_GDC_DISP_ATTR {
  MML_GDC_DISP_ATTR_OUTPUT_CONTROLLER = 0,
  MML_GDC_DISP_ATTR_X_RESOLUTION,
  MML_GDC_DISP_ATTR_Y_RESOLUTION,
  MML_GDC_DISP_ATTR_BUFF_ERR,
  MML_GDC_DISP_ATTR_BACKGROUND_COLOR
  }
- enum MML_GDC_DISP_WIN_ATTR {
  MML_GDC_DISP_WIN_ATTR_LAYER_ID = 0,
  MML_GDC_DISP_WIN_ATTR_SUB_LAYER_ID,
  MML_GDC_DISP_WIN_ATTR_TOPLEFT_X,
  MML_GDC_DISP_WIN_ATTR_TOPLEFT_Y,
  MML_GDC_DISP_WIN_ATTR_WIDTH,
  MML_GDC_DISP_WIN_ATTR_HEIGHT,
  MML_GDC_DISP_WIN_ATTR_SCREEN,
  MML_GDC_DISP_WIN_ATTR_COLOR,
  MML_GDC_DISP_WIN_ATTR_DISABLE,
  MML_GDC_DISP_WIN_ATTR_SWAP_INTERVAL,
  MML_GDC_DISP_WIN_ATTR_MAX_BUFFER,
  MML_GDC_DISP_WIN_ATTR_TILE_MODE,
  MML_GDC_DISP_WIN_ATTR_FEATURE
  }

Functions

Display Functions

- MM_ERROR mmlGdcDispOpenDisplay (MML_GDC_DISP_PROPERTIES *mode, MML_GDC_DISPLAY *display)
- MM_ERROR mmlGdcDispCloseDisplay (MML_GDC_DISPLAY display)
- MM_ERROR mmlGdcDispDitherCtrl (MML_GDC_DISPLAY display, MML_GDC_DISP_DITHER_ENABLE enable, MML_GDC_DISP_DITHER_MODE mode, MML_GDC_DISP_DITHER_RANGE range, MML_GDC_DISP_DITHER_FORMAT format)
- MM_ERROR mmlGdcDispCLUTData (MML_GDC_DISPLAY display, MML_GDC_DISP_CLUT_FORMAT format, const MM_S16 *pRed, const MM_S16 *pGreen, const MM_S16 *pBlue)
- MM_ERROR mmlGdcDispSyncVSync (MML_GDC_DISPLAY display, MML_GDC_SYNC sync, MM_S32 vsyncCnt)
- MM_ERROR mmlGdcDispSetAttribute (MML_GDC_DISPLAY display, MML_GDC_DISP_ATTR pname, MM_U32 param)
- MM_ERROR mmlGdcDispGetAttribute (MML_GDC_DISPLAY display, MML_GDC_DISP_ATTR pname, MM_U32 *pParam)
- MM_ERROR mmlGdcDispCommit (MML_GDC_DISPLAY display)

Window Functions

- MM_ERROR mmlGdcDispWinCreate (MML_GDC_DISPLAY display, MML_GDC_DISP_WINDOW_PROPERTIES *properties, MML_GDC_DISP_WINDOW *pWin)
- MM_ERROR mmlGdcDispWinDestroy (MML_GDC_DISP_WINDOW win)
- MM_ERROR mmlGdcDispWinSetSurface (MML_GDC_DISP_WINDOW win, MM_U32 target, MML_GDC_SURFACE surf)
- MM_ERROR mmlGdcDispWinSetBlendMode (MML_GDC_DISP_WINDOW win, MM_U32 blend_mode)
- MM_ERROR mmlGdcDispWinSetMatrix (MML_GDC_DISP_WINDOW win, MM_U32 target, const MM_FLOAT *matrix)
- MM_ERROR mmlGdcDispWinSync (MML_GDC_DISP_WINDOW win, MML_GDC_SYNC sync)
- MM_ERROR mmlGdcDispWinWaitSync (MML_GDC_DISP_WINDOW win, MML_GDC_SYNC sync)
- MM_ERROR mmlGdcDispWinSetAttribute (MML_GDC_DISP_WINDOW win, MML_GDC_DISP_WIN_ATTR pname, MM_U32 param)
- MM_ERROR mmlGdcDispWinGetAttribute (MML_GDC_DISP_WINDOW win, MML_GDC_DISP_WIN_ATTR pname, MM_U32 *pParam)
- MM_ERROR mmlGdcDispWinCommit (MML_GDC_DISP_WINDOW win)

13.10.1 Detailed Description

Display API.

13.11 mml_gdc_erp.h File Reference

Error Reporting API.

```
#include    "mm_defines.h"
#include "mm_gdc_erp.h"
```

Typedefs

- typedef void MM_PRINTFUNCTION (const char *string)

Functions

- MM_ERROR mmlGdcErpSetMessageLevel (MM_U32 moduleId, MM_ERP_MESSAGE_LEVEL level)
- MM_ERROR mmlGdcErpSetMessageChannel (MM_ERP_MESSAGE_DEST dest, MM_ERP_MESSAGE_CHANNEL_PROP prop)
- MM_ERROR mmlGdcErpSetBuffer (MM_ADDR bufferAddr, MM_U32 bufferSize)
- MM_ERROR mmlGdcErpSetPrintf (MM_PRINTFUNCTION *user_print_function)

13.11.1 Detailed Description

Error Reporting API.

13.12 mml_gdc_pixeng.h File Reference

Pixel Engine API.

```
#include "mml_gdc_sync.h"
#include "mm_types.h"
```

#include "mml_gdc_display.h"

#include "mm_gdc_errors.h"

Data Structures

- struct MML_GDC_PE_CONTEXT_CONTAINER

Macros

- #define MML_GDC_PE_API extern
- #define MML_GDC_PE_STORE 0x00000001U
- #define MML_GDC_PE_SRC 0x00000002U
- #define MML_GDC_PE_DST 0x00000004U
- #define MML_GDC_PE_MASK 0x00000008U
- #define MML_GDC_PE_ROP_BLACKNESS ((MM_U08)0x00)
- #define MML_GDC_PE_ROP_WHITENESS ((MM_U08)0xFF)
- #define MML_GDC_PE_ROP_SRCCOPY ((MM_U08)0xAA)
- #define MML_GDC_PE_ROP_NOTSRCCOPY ((MM_U08)0x55)
- #define MML_GDC_PE_ROP_MASKCOPY ((MM_U08)0xCC)
- #define MML_GDC_PE_ROP_NOTMASK ((MM_U08)0x33)
- #define MML_GDC_PE_ROP_MASKINVERT ((MM_U08)0x66)
- #define MML_GDC_PE_ROP_MSKAND ((MM_U08)0x88)
- #define MML_GDC_PE_ROP_MASKERASE ((MM_U08)0x22)
- #define MML_GDC_PE_ROP_NOTMASKERASE ((MM_U08)0x11)
- #define MML_GDC_PE_ROP_MERGEMASK ((MM_U08)0xEE)
- #define MML_GDC_PE_ROP_MERGEMASKNOT ((MM_U08)0xBB)
- #define MML_GDC_PE_ROP_DSTCOPY ((MM_U08)0xF0)
- #define MML_GDC_PE_ROP_NOTDSTCOPY ((MM_U08)0x0F)
- #define MML_GDC_PE_ROP_DSTPAINT ((MM_U08)0xFE)
- #define MML_GDC_PE_ROP_MASKSEL ((MM_U08)0xB8)
- #define MML_GDC_PE_ROP_DSTAND ((MM_U08)0x80)
- #define MML_GDC_PE_FILTER_NEAREST 0
- #define MML_GDC_PE_FILTER_BILINEAR 1
- #define MML_GDC_PE_ATTR_ZERO_TOP_LEFT 0U
- #define MML_GDC_PE_ATTR_ZERO_BOTTOM_LEFT 1U
- #define MML_GDC_PE_TILE_FILL_ZERO 0U
- #define MML_GDC_PE_TILE_FILL_CONSTANT 1U
- #define MML_GDC_PE_TILE_PAD 2U
- #define MML_GDC_PE_TILE_PAD_ZERO 3U

Typedefs

- typedef MML_GDC_PE_CONTEXT_CONTAINER * MML_GDC_PE_CONTEXT

Enumerations

- enum MML_GDC_PE_CTX_ATTR {
  MML_GDC_PE_CTX_ATTR_DITHER_COLOR,
  MML_GDC_PE_CTX_ATTR_DITHER_ALPHA,
  MML_GDC_PE_CTX_ATTR_DITHER_OFFSET,
  MML_GDC_PE_CTX_ATTR_FILTER,
  MML_GDC_PE_ATTR_ZERO_POINT
  }

- enum MML_GDC_PE_SURF_ATTR {
  MML_GDC_PE_SURF_ATTR_COLORMULTI,
  MML_GDC_PE_SURF_ATTR_ALPHAMULTI,
  MML_GDC_PE_SURF_ATTR_TILE_MODE,
  MML_GDC_PE_SURF_ATTR_USE_CLIPPING
  }
- enum MML_GDC_PE_BF {
  MML_GDC_PE_BF_GL_ZERO = 0x0U,
  MML_GDC_PE_BF_GL_ONE = 0x1U,
  MML_GDC_PE_BF_GL_SRC_COLOR = 0x300U,
  MML_GDC_PE_BF_GL_ONE_MINUS_SRC_COLOR = 0x301U,
  MML_GDC_PE_BF_GL_SRC_ALPHA = 0x302U,
  MML_GDC_PE_BF_GL_ONE_MINUS_SRC_ALPHA = 0x303U,
  MML_GDC_PE_BF_GL_DST_ALPHA = 0x304U,
  MML_GDC_PE_BF_GL_ONE_MINUS_DST_ALPHA = 0x305U,
  MML_GDC_PE_BF_GL_DST_COLOR = 0x306U,
  MML_GDC_PE_BF_GL_ONE_MINUS_DST_COLOR = 0x307U,
  MML_GDC_PE_BF_GL_SRC_ALPHA_SATURATE = 0x308U,
  MML_GDC_PE_BF_GL_CONSTANT_COLOR = 0x8001U,
  MML_GDC_PE_BF_GL_ONE_MINUS_CONSTANT_COLOR = 0x8002U,
  MML_GDC_PE_BF_GL_CONSTANT_ALPHA = 0x8003U,
  MML_GDC_PE_BF_GL_ONE_MINUS_CONSTANT_ALPHA = 0x8004U
  }
- enum MML_GDC_PE_BM {
  MML_GDC_PE_BM_GL_FUNC_ADD = 0x8006U,
  MML_GDC_PE_BM_GL_MIN = 0x8007U,
  MML_GDC_PE_BM_GL_MAX = 0x8008U,
  MML_GDC_PE_BM_GL_FUNC_SUBTRACT = 0x800AU,
  MML_GDC_PE_BM_GL_FUNC_REVERSE_SUBTRACT = 0x800BU,
  MML_GDC_PE_BM_VG_BLEND_SRC = 0x2000U,
  MML_GDC_PE_BM_VG_BLEND_SRC_OVER = 0x2001U,
  MML_GDC_PE_BM_VG_BLEND_DST_OVER = 0x2002U,
  MML_GDC_PE_BM_VG_BLEND_SRC_IN = 0x2003U,
  MML_GDC_PE_BM_VG_BLEND_DST_IN = 0x2004U,
  MML_GDC_PE_BM_VG_BLEND_MULTIPLY = 0x2005U,
  MML_GDC_PE_BM_VG_BLEND_SCREEN = 0x2006U,
  MML_GDC_PE_BM_VG_BLEND_DARKEN = 0x2007U,
  MML_GDC_PE_BM_VG_BLEND_LIGHTEN = 0x2008U,
  MML_GDC_PE_BM_VG_BLEND_ADDITIVE = 0x2009U
  }
- enum MML_GDC_PE_CMATRIX_FORMAT { MML_GDC_PE_CMATRIX_FORMAT_4X3 = 0 }
- enum MML_GDC_PE_GEO_MATRIX_FORMAT {
  MML_GDC_PE_GEO_MATRIX_FORMAT_3X2,
  MML_GDC_PE_GEO_MATRIX_FORMAT_3X3
  }
- enum MML_GDC_PE_CLUT_FORMAT {
  MML_GDC_PE_CLUT_FORMAT_33 = 33,
  MML_GDC_PE_CLUT_FORMAT_256 = 256
  }
- enum MML_GDC_PE_FILTER_CHANNEL {
  MML_GDC_PE_FILTER_CHANNEL_R = (1U<<3),
  MML_GDC_PE_FILTER_CHANNEL_G = (1U<<2),
  MML_GDC_PE_FILTER_CHANNEL_B = (1U<<1),

MML_GDC_PE_FILTER_CHANNEL_A = 1U,
MML_GDC_PE_FILTER_CHANNEL_RGB =
(MML_GDC_PE_FILTER_CHANNEL_R | MML_GDC_PE_FILTER_CHANNEL_G |
MML_GDC_PE_FILTER_CHANNEL_B),
MML_GDC_PE_FILTER_CHANNEL_RGBA =
(MML_GDC_PE_FILTER_CHANNEL_R | MML_GDC_PE_FILTER_CHANNEL_G |
MML_GDC_PE_FILTER_CHANNEL_B | MML_GDC_PE_FILTER_CHANNEL_A)
}

- enum MML_GDC_PE_FILTER_COLOR_FORMAT {
  MML_GDC_PE_FILTER_COLOR_FORMAT_R8G8B8,
  MML_GDC_PE_FILTER_COLOR_FORMAT_R5G6B5A8,
  MML_GDC_PE_FILTER_COLOR_FORMAT_R8G8B8A8,
  MML_GDC_PE_FILTER_COLOR_FORMAT_R10G10B10A8
  }

Functions

- MML_GDC_PE_API MM_ERROR mmlGdcPeResetContext (MML_GDC_PE_CONTEXT pectx)
- MML_GDC_PE_API MM_ERROR mmlGdcPeBindSurface (MML_GDC_PE_CONTEXT pectx, MM_U32 target, MML_GDC_SURFACE surface)
- MML_GDC_PE_API MM_ERROR mmlGdcPeAttribute (MML_GDC_PE_CONTEXT pectx, MML_GDC_PE_CTX_ATTR pname, MM_U32 param)
- MML_GDC_PE_API MM_ERROR mmlGdcPeColor (MML_GDC_PE_CONTEXT pectx, MM_U08 red, MM_U08 green, MM_U08 blue, MM_U08 alpha)
- MML_GDC_PE_API MM_ERROR mmlGdcPeSurfAttribute (MML_GDC_PE_CONTEXT pectx, MM_U32 target, MML_GDC_PE_SURF_ATTR pname, MM_U32 param)
- MML_GDC_PE_API MM_ERROR mmlGdcPeSurfColor (MML_GDC_PE_CONTEXT pectx, MM_U32 target, MM_U08 red, MM_U08 green, MM_U08 blue, MM_U08 alpha)
- MML_GDC_PE_API MM_ERROR mmlGdcPeBlendFunc (MML_GDC_PE_CONTEXT pectx, MML_GDC_PE_BF func_red_src, MML_GDC_PE_BF func_red_dst, MML_GDC_PE_BF func_green_src, MML_GDC_PE_BF func_green_dst, MML_GDC_PE_BF func_blue_src, MML_GDC_PE_BF func_blue_dst, MML_GDC_PE_BF func_alpha_src, MML_GDC_PE_BF func_alpha_dst)
- MML_GDC_PE_API MM_ERROR mmlGdcPeBlendMode (MML_GDC_PE_CONTEXT pectx, MML_GDC_PE_BM mode_red, MML_GDC_PE_BM mode_green, MML_GDC_PE_BM mode_blue, MML_GDC_PE_BM mode_alpha)
- MML_GDC_PE_API MM_ERROR mmlGdcPeRopOperation (MML_GDC_PE_CONTEXT pectx, MM_U08 op_red, MM_U08 op_green, MM_U08 op_blue, MM_U08 op_alpha)
- MML_GDC_PE_API MM_ERROR mmlGdcPeSetMatrix (MML_GDC_PE_CONTEXT pectx, MM_U32 target, MML_GDC_PE_GEO_MATRIX_FORMAT format, const MM_FLOAT *fMatrix)
- MML_GDC_PE_API MM_ERROR mmlGdcPeCLUTData (MML_GDC_PE_CONTEXT pectx, MML_GDC_PE_CLUT_FORMAT format, const MM_S16 *pRed, const MM_S16 *pGreen, const MM_S16 *pBlue)
- MML_GDC_PE_API MM_ERROR mmlGdcPeColorMatrix (MML_GDC_PE_CONTEXT pectx, MML_GDC_PE_CMATRIX_FORMAT format, const MM_FLOAT *fMatrix)
- MML_GDC_PE_API MM_ERROR mmlGdcPeGetDrawBox (MML_GDC_PE_CONTEXT pectx, MM_U32 *x,MM_U32 *y, MM_U32 *w, MM_U32 *h, MM_U32 reset)
- MML_GDC_PE_API MM_ERROR mmlGdcPeActiveArea (MML_GDC_PE_CONTEXT pectx, MM_U32 target, MM_S32 x, MM_S32 y, MM_U32 w, MM_U32 h)
- MML_GDC_PE_API MM_ERROR mmlGdcPeSelectArea (MML_GDC_PE_CONTEXT pectx, MM_U32 target)
- MML_GDC_PE_API MM_ERROR mmlGdcPeFill (MML_GDC_PE_CONTEXT pectx, MM_U32 x, MM_U32 y,MM_U32 w, MM_U32 h)

- MML_GDC_PE_API MM_ERROR mmlGdcPeBlt (MML_GDC_PE_CONTEXT pectx, MM_FLOAT offsetx, MM_FLOAT offsety)
- MML_GDC_PE_API MM_ERROR mmlGdcPeFinish (void)
- MML_GDC_PE_API MM_ERROR mmlGdcPeFlush (void)
- MML_GDC_PE_API MM_ERROR mmlGdcPeSync (MML_GDC_SYNC sync)
- MML_GDC_PE_API MM_ERROR mmlGdcPeWaitSync (MML_GDC_SYNC sync)
- MM_ERROR mmlGdcPeWaitForDispFrameEnd (MML_GDC_DISPLAY display, MM_U32 line)

13.12.1 Detailed Description

Pixel Engine API.

13.13 mml_gdc_surfman.h File Reference

Surface Manager Interface.

#include "mm_types.h"

#include "mm_gdc_module_id.h"

#include "mm_gdc_errors.h"

Data Structures

- struct MML_GDC_SURFACE_CONTAINER

Macros

- #define MML_GDC_SURFACE_MAX_WIDTH 4096
- #define MML_GDC_SURFACE_MAX_HEIGHT 4096
- #define MML_GDC_SURFACE_CONTROL_WIDTH 2048
- #define MML_GDC_SURFACE_CONTROL_HEIGHT 2048

Typedefs

- typedef MML_GDC_SURFACE_CONTAINER * MML_GDC_SURFACE

Enumerations

- enum MML_GDC_SURF_FORMAT {
  MML_GDC_SURF_FORMAT_R8G8B8A8 = 0x00,
  MML_GDC_SURF_FORMAT_A8B8G8R8,
  MML_GDC_SURF_FORMAT_A8R8G8B8,
  MML_GDC_SURF_FORMAT_B8G8R8A8,
  MML_GDC_SURF_FORMAT_R8G8B8X8,
  MML_GDC_SURF_FORMAT_X8B8G8R8,
  MML_GDC_SURF_FORMAT_X8R8G8B8,
  MML_GDC_SURF_FORMAT_R8G8B8,
  MML_GDC_SURF_FORMAT_B8G8R8,
  MML_GDC_SURF_FORMAT_R6G6B6,
  MML_GDC_SURF_FORMAT_R4G4B4A4,
  MML_GDC_SURF_FORMAT_A4R4G4B4,
  MML_GDC_SURF_FORMAT_R5G5B5A1,
  MML_GDC_SURF_FORMAT_A1R5G5B5,
  MML_GDC_SURF_FORMAT_A1B5G5R5,

```
        MML_GDC_SURF_FORMAT_B5G5R5A1,
        MML_GDC_SURF_FORMAT_R5G6B5,
        MML_GDC_SURF_FORMAT_A8RGB8,
        MML_GDC_SURF_FORMAT_RGB8,
        MML_GDC_SURF_FORMAT_A8,
        MML_GDC_SURF_FORMAT_A4RGB4,
        MML_GDC_SURF_FORMAT_A4,
        MML_GDC_SURF_FORMAT_A2,
        MML_GDC_SURF_FORMAT_A1,
        MML_GDC_SURF_FORMAT_RGB1
    }
- enum MML_GDC_SURF_COMP {
        MML_GDC_SURF_COMP_NON = 0x4,
        MML_GDC_SURF_COMP_RLC = 0x3,
        MML_GDC_SURF_COMP_RLA = 0x2,
        MML_GDC_SURF_COMP_RLAD = 0x0
    }
- enum MML_GDC_SURF_CLF {
        MML_GDC_SURF_CLF_R8G8B8,
        MML_GDC_SURF_CLF_B8G8R8,
        MML_GDC_SURF_CLF_R5G5B5,
        MML_GDC_SURF_CLF_A1R5G5B5,
        MML_GDC_SURF_CLF_A4R4G4B4
    }
- enum MML_GDC_SURF_CLM {
        MML_GDC_SURF_CLM_NEUTRAL = 0x0,
        MML_GDC_SURF_CLM_INDEX_RGB,
        MML_GDC_SURF_CLM_INDEX_RGBA
    }
- enum MML_GDC_SURF_ATTR {
        MML_GDC_SURF_ATTR_BASE_ADDRESS = 0x0,
        MML_GDC_SURF_ATTR_PHYS_ADDRESS,
        MML_GDC_SURF_ATTR_BASE_ADDRESS2,
        MML_GDC_SURF_ATTR_PHYS_ADDRESS2,
        MML_GDC_SURF_ATTR_WIDTH,
        MML_GDC_SURF_ATTR_HEIGHT,
        MML_GDC_SURF_ATTR_STRIDE,
        MML_GDC_SURF_ATTR_BITPERPIXEL,
        MML_GDC_SURF_ATTR_COLORBITS,
        MML_GDC_SURF_ATTR_COLORSHIFT,
        MML_GDC_SURF_ATTR_COMPRESSION_FORMAT,
        MML_GDC_SURF_ATTR_RLAD_MAXCOLORBITS,
        MML_GDC_SURF_ATTR_SIZEINBYTES,
        MML_GDC_SURF_ATTR_CLUTMODE,
        MML_GDC_SURF_ATTR_CLUTCOUNT,
        MML_GDC_SURF_ATTR_CLUTBITPERPIXEL,
        MML_GDC_SURF_ATTR_CLUTCOLORBITS,
        MML_GDC_SURF_ATTR_CLUTCOLORSHIFT,
        MML_GDC_SURF_ATTR_CLUTBUFFERADDRESS,
        MML_GDC_SURF_ATTR_CLUTBUFFER_PHYS_ADDRESS,
        MML_GDC_SURF_ATTR_SURF_FORMAT,
        MML_GDC_SURF_ATTR_USERDEFINED
    }
```

Functions

- MM_ERROR mmlGdcSmResetSurfaceObject (MML_GDC_SURFACE surf)
- MM_ERROR mmlGdcSmAssignBuffer (MML_GDC_SURFACE surf, MM_U32 uWidth, MM_U32 uHeight, MML_GDC_SURF_FORMAT eFormat, void *pBufferAddress, MM_U32 uRleWords)
- MM_ERROR mmlGdcSmAssignClut (MML_GDC_SURFACE surf, MML_GDC_SURF_CLM eMode, MM_U32 uCount, MML_GDC_SURF_CLF eFormat, void *pBufferAddress)
- MM_ERROR mmlGdcSmSetAttribute (const MML_GDC_SURFACE surf, MML_GDC_SURF_ATTR eName, MM_U32 uValue)
- MM_ERROR mmlGdcSmGetAttribute (const MML_GDC_SURFACE surf, MML_GDC_SURF_ATTR eName, MM_U32 *puValue)

13.13.1 Detailed Description

Surface Manager Interface.

13.14 mml_gdc_sync.h File Reference

Synchronization of framebuffer operations.

#include "mm_gdc_errors.h"

Data Structures

- struct MML_GDC_SYNC_CONTAINER

Typedefs

- typedef MML_GDC_SYNC_CONTAINER * MML_GDC_SYNC

Functions

- MM_ERROR mmlGdcSyncReset (MML_GDC_SYNC sync)
- MM_ERROR mmlGdcSyncWait (MML_GDC_SYNC sync, MM_S32 timeout)
- MM_ERROR mmlGdcSyncIncr (MML_GDC_SYNC sync, MM_S32 incr)

13.14.1 Detailed Description

Synchronization of framebuffer operations.

13.15 mml_gdc_sysinit.h File Reference

Driver Initialization Module.

#include "mm_gdc_errors.h"

Data Structures

- struct MML_GDC_SYSINIT_INFO

Macros

- #define GFX_PLL_MIN 20000000U
- #define GFX_PLL_MAX 415000000U

Default initializer

- #define MML_GDC_SYSINIT_INITIALIZER

Resource names

- #define MM_GDC_RES_DISP0 (1U << 0U)
- #define MM_GDC_RES_LAYER0 (1U << 1U)
- #define MM_GDC_RES_LAYER1 (1U << 2U)
- #define MM_GDC_RES_FETCH_DECODE0 (1U << 3U)
- #define MM_GDC_RES_FETCH_LAYER0 (1U << 4U)

Functions

- MM_ERROR mmlGdcSysInitializeDriver (MML_GDC_SYSINIT_INFO *pDriverInitInfo)
- MM_ERROR mmlGdcSysUninitializeDriver (void)
- MM_ERROR mmlGdcSysSetInstructionBuffer (void *address, MM_U32 size)

13.15.1 Detailed Description

Driver Initialization Module.

13.16 pe_matrix.h File Reference

Provide some matrix utility functions.

#include "mm_types.h"

Macros

- #define MML_GDC_2D_MATRIX_API extern

Typedefs

- typedef MM_FLOAT Mat3x2 [6]
- typedef MM_FLOAT Mat3x3 [9]
- typedef MM_FLOAT Mat4x4 [16]
- typedef MM_FLOAT Mat4x3 [12]
- typedef MM_FLOAT Mat5x4 [20]

Functions

Matrix functions for geometric operations

- MML_GDC_2D_MATRIX_API void utMat3x2Copy (Mat3x2 dst, const Mat3x2 src)
- MML_GDC_2D_MATRIX_API void utMat3x2Multiply (Mat3x2 dst, const Mat3x2 src1, const Mat3x2 src2)
- MML_GDC_2D_MATRIX_API void utMat3x2LoadIdentity (Mat3x2 m)
- MML_GDC_2D_MATRIX_API void utMat3x2Translate (Mat3x2 m, MM_FLOAT x, MM_FLOAT y)

- MML_GDC_2D_MATRIX_API void utMat3x2TranslatePre (Mat3x2 m, MM_FLOAT x, MM_FLOAT y)
- MML_GDC_2D_MATRIX_API void utMat3x2Scale (Mat3x2 m, MM_FLOAT x, MM_FLOAT y)
- MML_GDC_2D_MATRIX_API void utMat3x2ScalePre (Mat3x2 m, MM_FLOAT x, MM_FLOAT y)
- MML_GDC_2D_MATRIX_API void utMat3x2Rot (Mat3x2 m, MM_FLOAT f)
- MML_GDC_2D_MATRIX_API void utMat3x2RotPre (Mat3x2 m, MM_FLOAT f)
- MML_GDC_2D_MATRIX_API MM_U32 utMat3x2Invert (Mat3x2 m)
- MML_GDC_2D_MATRIX_API void utMat3x2GetXY (const Mat3x2 m, const MM_FLOAT x, const MM_FLOAT y, MM_FLOAT *xout, MM_FLOAT *yout)
- MML_GDC_2D_MATRIX_API void utMat3x3LoadIdentity (Mat3x3 m)
- MML_GDC_2D_MATRIX_API void utMat3x3Copy (Mat3x3 dst, const Mat3x3 src)
- MML_GDC_2D_MATRIX_API void utMat3x3Multiply (Mat3x3 dst, const Mat3x3 src1, const Mat3x3 src2)
- MML_GDC_2D_MATRIX_API void utMat3x3Translate (Mat3x3 m, MM_FLOAT x, MM_FLOAT y)
- MML_GDC_2D_MATRIX_API void utMat3x3TranslatePre (Mat3x3 m, MM_FLOAT x, MM_FLOAT y)
- MML_GDC_2D_MATRIX_API void utMat3x3Scale (Mat3x3 m, MM_FLOAT x, MM_FLOAT y)
- MML_GDC_2D_MATRIX_API void utMat3x3RotX (Mat3x3 m, MM_FLOAT f)
- MML_GDC_2D_MATRIX_API void utMat3x3RotZ (Mat3x3 m, MM_FLOAT f)
- MML_GDC_2D_MATRIX_API void utMat4x4Copy (Mat4x4 dst, const Mat4x4 src)
- MML_GDC_2D_MATRIX_API void utMat4x4Multiply (Mat4x4 dst, const Mat4x4 src1, const Mat4x4 src2)
- MML_GDC_2D_MATRIX_API void utMat4x4LoadIdentity (Mat4x4 m)
- MML_GDC_2D_MATRIX_API void utMat4x4Translate (Mat4x4 m, MM_FLOAT x, MM_FLOAT y, MM_FLOAT z)
- MML_GDC_2D_MATRIX_API void utMat4x4Scale (Mat4x4 m, MM_FLOAT x, MM_FLOAT y, MM_FLOATz)
- MML_GDC_2D_MATRIX_API void utMat4x4RotX (Mat4x4 m, MM_FLOAT f)
- MML_GDC_2D_MATRIX_API void utMat4x4RotY (Mat4x4 m, MM_FLOAT f)
- MML_GDC_2D_MATRIX_API void utMat4x4RotZ (Mat4x4 m, MM_FLOAT f)
- MML_GDC_2D_MATRIX_API void utMat4x4Perspective (Mat4x4 m, MM_FLOAT fovy, MM_FLOAT aspect, MM_FLOAT zNear, MM_FLOAT zFar)
- MML_GDC_2D_MATRIX_API void utMat4x4GetXYZ (Mat4x4 m, float x, float y, float z, float *xout, float *yout, float *zout)

Matrix functions for the conversion of matrices

- MML_GDC_2D_MATRIX_API void utMat3x2ToMat4x4 (Mat3x2 src, Mat4x4 dst)
- MML_GDC_2D_MATRIX_API void utMat3x3ToMat4x4 (Mat3x3 src, Mat4x4 dst)
- MML_GDC_2D_MATRIX_API void utMat4x4ToMat3x3 (Mat4x4 src, Mat3x3 dst)
- MML_GDC_2D_MATRIX_API void utMat4x4ToMat3x2 (Mat4x4 src, Mat3x2 dst)

Matrix functions for color operations

- MML_GDC_2D_MATRIX_API void utMat4x3Copy (Mat4x3 dst, const Mat4x3 src)
- MML_GDC_2D_MATRIX_API void utMat4x3Multiply (Mat4x3 dst, const Mat4x3 src1, const Mat4x3 src2)
- MML_GDC_2D_MATRIX_API void utMat4x3LoadIdentity (Mat4x3 m)
- MML_GDC_2D_MATRIX_API void utMat5x4LoadIdentity (Mat5x4 m)
- MML_GDC_2D_MATRIX_API void utMat4x3CalcColMatrix (Mat4x3 dst, MM_FLOAT fContrast, MM_FLOAT fBrightness, MM_FLOAT fSaturation, MM_FLOAT fHue)

13.16.1 Detailed Description

Provide some matrix utility functions.

13.17 sm_util.h File Reference

This is just a helper implementation for development and will be removed in the final version.

#include <stdio.h>

#include "mml_gdc_surfman.h"

Macros

- #define UTIL_SUCCESS(rc, execute)
- #define UTIL_ERR_OUT_OF_MEMORY MM_ERRCODE(0x31000001)

Functions

- MM_ERROR utSurfReadBitmap (MML_GDC_SURFACE surface, void **pImage, MM_U32 *baseAddr, MM_U32 *clutAddr)
- MM_ERROR utSurfLoadBitmap (MML_GDC_SURFACE surface, const void *pImage, MM_BOOL bCopyToRAM)
- MM_S32 utSurfWidth (MML_GDC_SURFACE surf)
- MM_S32 utSurfHeight (MML_GDC_SURFACE surf)
- MM_ERROR utSurfCreateBuffer (MML_GDC_SURFACE surf, MM_U32 w, MM_U32 h, MML_GDC_SURF_FORMAT eFormat)
- void utSurfDeleteBuffer (MML_GDC_SURFACE surf)
- MM_ERROR utSurfGetPixel (MML_GDC_SURFACE src, MM_U32 x, MM_U32 y, MM_U08 *r, MM_U08 *g, MM_U08 *b, MM_U08 *a)
- MM_ERROR utSurfSetPixel (MML_GDC_SURFACE src, MM_U32 x, MM_U32 y, MM_U08 r, MM_U08 g, MM_U08 b, MM_U08 a)

13.17.1 Detailed Description

This is just a helper implementation for development and will be removed in the final version.

13.18 ut_class_ctx.h File Reference

This class abstracts an MML_GDC_PE_CONTEXT.

#include "mml_gdc_pixeng.h"

#include "ut_compatibility.h"

Data Structures

- class CCtx

13.18.1 Detailed Description

This class abstracts an MML_GDC_PE_CONTEXT.

13.19 ut_class_device.h File Reference

This class abstracts the device initialization.

#include "mml_gdc_display.h"

#include "mml_gdc_sysinit.h"

#include "sm_util.h"

#include "ut_compatibility.h"

#include "ut_memman.h"

Data Structures

- class CDevice

13.19.1 Detailed Description

This class abstracts the device initialization.

13.20 ut_class_display.h File Reference

This class abstracts the display initialisation.

#include "mml_gdc_display.h"

#include "mml_gdc_sysinit.h"

#include "sm_util.h"

Data Structures

- class CDisplay

13.20.1 Detailed Description

This class abstracts the display initialisation.

13.21 ut_class_menu.h File Reference

This class realizes a simple menu.

#include "wchar.h"

#include "sm_util.h"

#include "ut_compatibility.h"

#include "ut_class_window.h"

#include "ut_class_surface.h"

#include "ut_freetype.h"

#include "pe_matrix.h"

Data Structures

- class CMenuItem
- class CMenu

13.21.1 Detailed Description

This class realizes a simple menu.

13.22 ut_class_rlad.h File Reference

This sample code can be used to compress a buffer using the MML_GDC_SURF_COMP_RLA, MML_GDC_SURF_COMP_RLAD or ::MML_GDC_SURF_COMP_RLAD_UNIFORM format.

#include <assert.h>

#include <vector>

#include <queue>

Data Structures

- class RLAD
- class RLAD::Frame
- struct RLAD::Frame::Pixel
- class RLAD::BitStream
- struct RLAD::Package

Macros

- #define RLAD_VERSION 1.02

13.22.1 Detailed Description

This sample code can be used to compress a buffer using the MML_GDC_SURF_COMP_RLA, MML_GDC_SURF_COMP_RLAD or ::MML_GDC_SURF_COMP_RLAD_UNIFORM format.

13.22.2 Macro Definition Documentation

13.22.2.1 #define RLAD_VERSION 1.02

Version information of this file

13.23 ut_class_surface.h File Reference

This class abstracts MML_GDC_SURFACE objects.

#include <stdio.h>

#include <string.h>

#include "mml_gdc_surfman.h"

#include "sm_util.h"

Data Structures

- class CSurface< NUM_BUFFERS >

13.23.1 Detailed Description

This class abstracts MML_GDC_SURFACE objects.

13.24 ut_class_window.h File Reference

This class abstracts windows.

```
#include "mml_gdc_display.h"
#include "mml_gdc_sysinit.h"
#include   "ut_class_display.h"
#include "ut_class_surface.h"
#include   "ut_class_ctx.h"
#include "sm_util.h"
#include "dbg_win.h"
```

Data Structures

- class CWindow
- class CSurfaceWindow< NUM_BUFFERS >
- class CStaticSurfaceWindow

13.24.1 Detailed Description

This class abstracts windows.

13.25 ut_compatibility.h File Reference

This file defines some interfaces that are part of other drivers. The util library implements very simple instances of it but they must be not used for software products. However it allows to run the sample applications.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "mml_gdc_surfman.h"
#include   "mml_gdc_pixeng.h"
```

Enumerations

- enum UTIL_VRAM_CONFIG {
  UTIL_VRAM_CONFIG_VRAM_ONLY = 0x1U,
  UTIL_VRAM_CONFIG_SDRAM_ONLY = 0x2U,
  UTIL_VRAM_CONFIG_VRAM_PREFERRED = 0x3U
  }

Functions

- MM_ERROR mmlGdcSmGenSurfaceObjects (MM_U32 uCnt, MML_GDC_SURFACE *pSurfaces)

- MM_ERROR mmlGdcSmDeleteSurfaceObjects (MM_U32 uCnt, MML_GDC_SURFACE *pSurfaces)
- MM_ERROR mmlGdcPeGenContext (MML_GDC_PE_CONTEXT *pPectx)
- void mmlGdcPeDeleteContext (MML_GDC_PE_CONTEXT pectx)
- void * mmlOsLibcMalloc (size_t _Size)
- void mmlOsLibcFree (void *_Memory)
- MM_ERROR mmlGdcVideoConfig (UTIL_VRAM_CONFIG config)
- void * mmlGdcVideoAlloc (MM_U32 size, MM_U32 alignment, MM_ADDR *pAddr)
- void mmlGdcVideoFree (void *addr)
- MM_ERROR mmlGdcVideoGetSize (MM_U32 *size)
- MM_ERROR mmlGdcVideoGetFreeTotal (MM_U32 *size)
- MM_ERROR mmlGdcVideoGetLargestBlock (MM_U32 *size)
- MM_ERROR mmlGdcSyncCreate (MM_U32 uCnt, MML_GDC_SYNC *pSyncObjects)
- MM_ERROR mmlGdcSyncDelete (MM_U32 uCnt, MML_GDC_SYNC *pSyncObjects)

13.25.1 Detailed Description

This file defines some interfaces that are part of other drivers. The util library implements very simple instances of it but they must be not used for software products. However it allows to run the sample applications.

13.26 ut_compression.h File Reference

This file defines a helper function that can be used to compress a surface.

#include "mml_gdc_surfman.h"

Functions

- MM_ERROR utSurfCompress (MML_GDC_SURFACE surf, MML_GDC_SURF_COMP mode)

13.26.1 Detailed Description

This file defines a helper function that can be used to compress a surface.

13.27 ut_memman.h File Reference

This file defines some interfaces for the memory management.

#include "mm_defines.h"

Macros

- #define MML_ERR_MMAN_INVALID_PARAMETER MM_ERRCODE(0x18010001)
- #define MML_ERR_MMAN_NO_MEMORY MM_ERRCODE(0x18010002)
- #define MML_ERR_MMAN_NO_VRAM MM_ERRCODE(0x18010003)
- #define MML_ERR_MMAN_INVALID_MEMORY MM_ERRCODE(0x18010004)
- #define MML_ERR_MMAN_ACCESS_FAILED MM_ERRCODE(0x18010005)
- #define MM_VRAM_BASE 0xD0000000U
- #define MM_VRAM_SIZE 0x00080000U
- #define MM_SDRAM_BASE 0xB0080000U
- #define MM_SDRAM_SIZE 0x01000000U

Typedefs

- typedef void * MML_MMAN_HEAP_HANDLE

Functions

- MM_ERROR utMmanReset (void)
- MM_ERROR utMmanCreateHeap (MML_MMAN_HEAP_HANDLE *hdlmem, MM_U32 size, MM_U32 base-Address)
- MM_ERROR utMmanDestroyHeap (MML_MMAN_HEAP_HANDLE hdlmem)
- MM_ERROR utMmanHeapAlloc (MML_MMAN_HEAP_HANDLE hdlmem, MM_U32 size, MM_U32 alignment, MM_ADDR *addr)
- MM_ERROR utMmanHeapFree (MML_MMAN_HEAP_HANDLE hdlmem, void *addr)
- MM_ERROR utMmanGetSize (MML_MMAN_HEAP_HANDLE hdlmem, MM_U32 *size)
- MM_ERROR utMmanGetFree (MML_MMAN_HEAP_HANDLE hdlmem, MM_U32 *size)
- MM_ERROR utMmanGetLargest (MML_MMAN_HEAP_HANDLE hdlmem, MM_U32 *size)

13.27.1 Detailed Description

This file defines some interfaces for the memory management.

13.28 ut_rlc.h File Reference

This sample code can be used to create a run-length encoded buffer.

```
#include "mm_types.h"
#include    "mm_defines.h"
```

Functions

- MM_U32 utRldEncode (MM_U32 *pixeldata, MM_U32 unWidth, MM_U32 unHeight, MM_U32 strideBytes, MM_U32 dataBpp, MM_U32 *rld, MM_U32 rldCount)

13.28.1 Detailed Description

This sample code can be used to create a run-length encoded buffer.

14. Major Changes

Page	Section	Change results
Revision 1.0		
-	-	Initial release

AN709-00022-1v0-E

Cypress • Application Note

FM3 Family
32-BIT MICROCONTROLLER
S6E2DH/S6E2DF/S6E2D5/S6E2D3 Series
GRAPHIC DRIVER USER MANUAL

September 2015 Rev. 1.0

Published: Cypress Semiconductor Corp.
Edited: Communications Dept.

Colophon

The products described in this document are designed, developed and manufactured as contemplated for general use, including without limitation, ordinary industrial use, general office use, personal use, and household use, but are not designed, developed and manufactured as contemplated (1) for any use that includes fatal risks or dangers that, unless extremely high safety is secured, could have a serious effect to the public, and could lead directly to death, personal injury, severe physical damage or other loss (i.e., nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system), or (2) for any use where chance of failure is intolerable (i.e., submersible repeater and artificial satellite). Please note that Cypress will not be liable to you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products. Any semiconductor devices have an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions. If any products described in this document represent goods or technologies subject to certain restrictions on export under the Foreign Exchange and Foreign Trade Law of Japan, the US Export Administration Regulations or the applicable laws of any other country, the prior authorization by the respective government entity will be required for export of those products.

Trademarks and Notice

The contents of this document are subject to change without notice. This document may contain information on a Cypress product under development by Cypress. Cypress reserves the right to change or discontinue work on any product without notice. The information in this document is provided as is without warranty or guarantee of any kind as to its accuracy, completeness, operability, fitness for particular purpose, merchantability, non-infringement of third-party rights, or any other warranty, express, implied, or statutory. Cypress assumes no liability for any damages of any kind arising out of the use of the information in this document.