ORSoC Graphics accelerator Specification

Per Lenander, Anton Fosselius

June 1, 2012

Revision history

Rev.	Date	Author	Description
0.1.0	23/3/2012	Per Lenander	Initial draft

Contents

1 Introduction

The ORSoC Graphics accelerator allows the user to do advanced vector rendering and 2D blitting to a memory area. The core supports operations such as drawing textures, lines, curves and filling rectangular and triangular areas with color.

This IP Core is designed to integrate with the OpenRISC processor through a Wishbone bus interface. The core itself has no means of displaying the information rendered, for this purpose it can work alongside a display component, such as the enhanced VGA/LCD IP core found on OpenCores.

1.1 Features

- 32-bit Wishbone bus interface
- Integrates with enhanced VGA/LCD IP core
- Support for 16 bit color depth
- Support for variable resolution
- Acceleration of line operations
- Acceleration of rectangle and triangle rasterization
- Acceleration of memory copy operations
- Textures can be saved to video memory
- Vector transformation and rasterization
- Clipping/Scissoring
- Alpha blending and colorkeying
- Filled Bezier curves
- Bitmap Fonts
- Vector Fonts (ttf)
- Interpolation of colors
- UV-Mapping
- Transformation (scaling and rotation)
- 3D model support (3d degree .obj)
- Z-Buffer (triangles drawn in order of appearance)
- Requires 10000 Slice LUTs (Xilinx ISE 13.4)

1.2 IP Core directory structure

A basic overview of the contents of the IP core source folder can be found in figure 1. The **rtl** folder also contains files for implementing the component in ORPSoCv2.

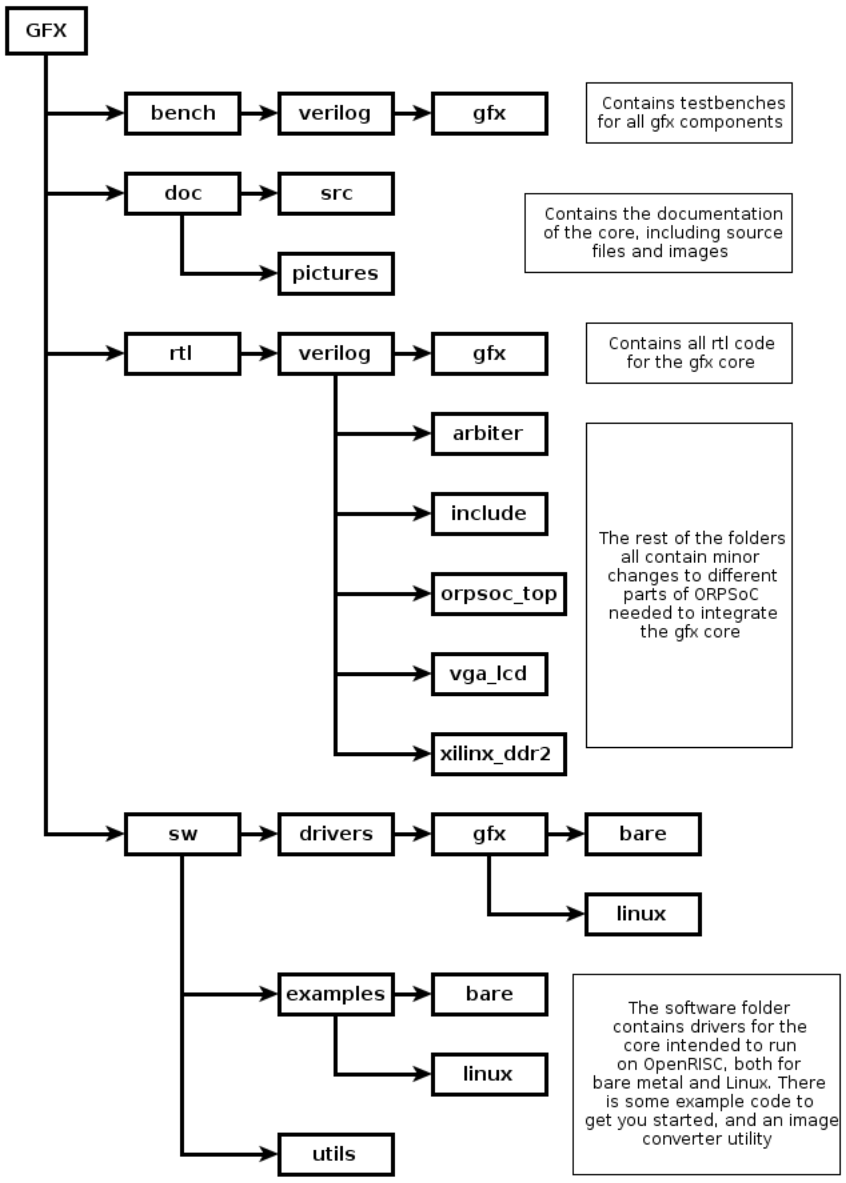
GFX
bench
verilog
gfx
Contains testbenches for all gfx components
doc
src
pictures
Contains the documentation of the core, including source files and images
rtl
verilog
gfx
Contains all rtl code for the gfx core
arbiter
include
orpsoc_top
vga_lcd
xilinx_ddr2
The rest of the folders all contain minor changes to different parts of ORPSoC needed to integrate the gfx core
sw
drivers
gfx
bare
linux
examples
bare
linux
utils
The software folder contains drivers for the core intended to run on OpenRISC, both for bare metal and Linux. There is some example code to get you started, and an image converter utility

Figure 1: Directory structure of the ORSoC graphics accelerator.

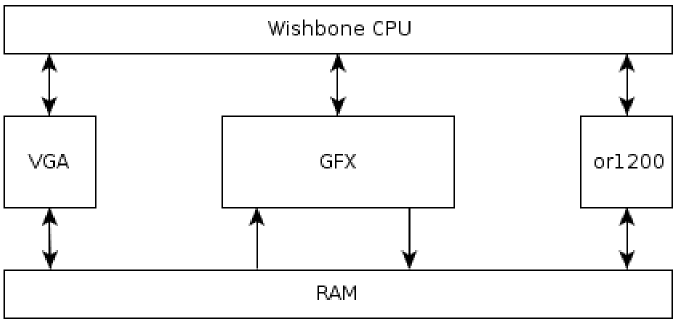

Figure 2: Overview of the ORPSoCv2's wishbone interconnection.

2 Architecture

2.1 Overview

A topology of how the orgfx is connected to the VGA driver and the OpenRisc core is shown in figure 2. The orgfx has three wishbone interfaces: one read-/write port that is used to communicate with the host CPU. One read port that reads texture/alpha blending information from the RAM and one write port to write pixel information to the RAM.

2.2 Concepts

This section describes a few basic terms used in this document.

Video memory – The orgfx component writes pixels one by one to an external memory, usually a SDRAM or DDR RAM chip. The CPU should also have access to this memory space to be able to write pixels directly (the easiest way to load textures).

Render target – The render target, defined by the target base and size registers, describes the area to which all operations render pixels. It is possible to change the base address and size, enabling render-to-texture and double buffering.

Surface/Texture – Any memory area that can be rendered to, including the render target, is considered a surface. A surface is defined by it's base address and size. There are two main surfaces that the orgfx device handles: the render target and the currently active texture. Swapping between different textures has to be done in software. The operation of setting the current render target or texture is referred to as *binding.*

Source, Destination and Clip rectangles – There are three sets of rectangles that affect rendering, each described by two points. The first point sets the beginning of the rectangle, while the second point sets the pixel after the end of the rectangle. This way, a rectangle exactly filling the screen would be (0,0,640,480) at 640x480 resolution. See figure 3;

Source rectangle – The source rectangle defines what pixels should be read from a texture during textured operations. The points are defined in the

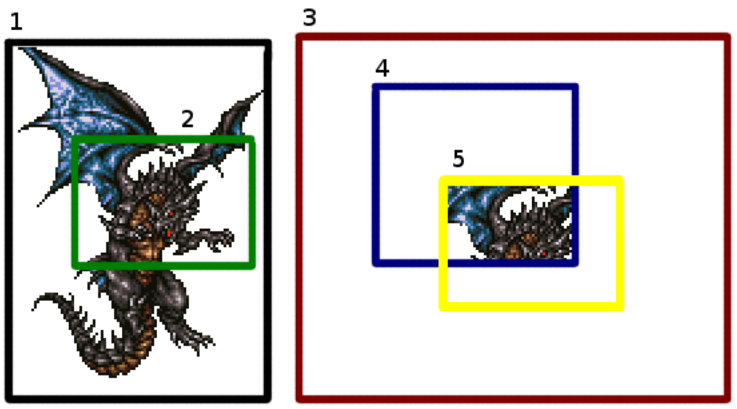

Figure 3: 1. Texture, 2. Source, 3. Render target, 4. Clip, 5. Destination

coordinates of the currently bound texture. This way sections of a texture can be drawn (good for tile maps or bitmap fonts).

Destination rectangle – The destination rectangle defines where operations such as draw pixel and draw line will draw pixels, in the coordinates of the render target.

Clip rectangle – The clip rectangle defines an area within the current render target which is valid to draw to. Any pixels outside this rectangle are discarded in the rasterization step. Pixels outside of the render target are automatically discarded.

2.3 Coordinate precision

The orgfx core supports variable coordinate precision through two parameters, **point_width** and **subpixel_width**. Both parameters defaults to 16 bits width.

Target size, clip and source rects are defined as **point_width** bit integers. Destination points are defined as fixed point numbers, with **point_width** bit integer precision and **subpixel_width** fractional precision. Internally many calculations are done with fixed point logic.

2.4 Instruction FIFO

All wishbone writes sent to the slave interface will pass through an instruction fifo. If the device is in the busy state (when the pipeline is active) the instruction will be queued instead of being set immediately. This is important to take into account when reading from registers, since an operation that changes the register being read might be queued. To find out if the device is busy, poll the status register and check if the busy bit is high.

2.5 Pipeline

The orgfx core uses a pipelined architecture to speed up operation. An overview of the pipeline can be seen in figure 4. Each module in the pipeline communicates with acknowledge and write signals. A module will not assert write to the next

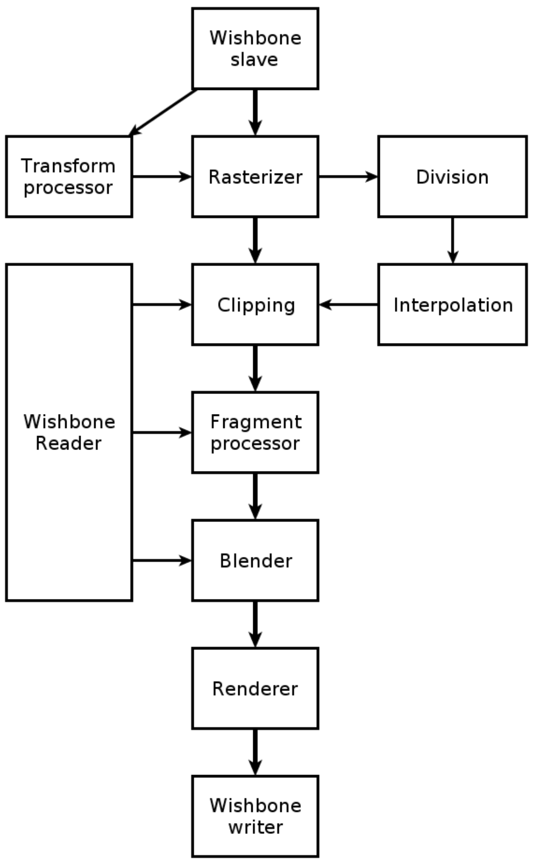

Figure 4: Picture of the orgfx pipeline

module unless it receives an ack first (or if the module was previously in a ready state, in which case the downstream pipeline is empty). All ack and write signals are always exactly one clock tick long, to prevent triggering multiple instances of the same instruction.

Each module in the pipeline may hold the upstream pipeline for several clock ticks. For example, the rasterizer will prevent incoming raster instructions until all the pixels for the current operation are generated. When the rasterizer is ready for new data, it will send an ack upstream.

2.6 Description of core modules

2.6.1 Wishbone slave

The wishbone slave handles all communication from the main OpenRISC processor (or other master cpu). This component holds all the registers, and the instruction fifo that sets them. This component can be in one of two states: *busy* or *wait*. It enters the busy state when a pipeline operation is initialized, and returns to the wait state when the operation is finished.

2.6.2 Transformation processor

The transformation processor handles rotations and scaling.

2.6.3 Vector processor

This module generates the Bézier curve and can be skipped if no vector graphics is drawn.

2.6.4 Rasterizer

This rasterizer generates pixel coordinates from points for several different operations.

2.6.5 Clipper

Discard generated pixel if clipping is enabled and pixel is out of bounds. Always discard pixels outside of the target area.

2.6.6 Fragment processor

The fragment processor adds color to the pixel generated by the rasterizer. If texturing is disabled, a color supplied from the color register is used. If texturing is enabled on the other hand, the u v coordinates supplied by the rasterizer are used to fetch a pixel from the active texture. If colorkeying is enabled and the fetched color matches the color key, the current pixel is discarded.

2.6.7 Blender

The blender module performs alpha blending if this is enabled. The module fetches the color of the pixel that the current operation will write to, and the mixes the value of the target color and the color from the fragment processor using the following algorithm:

$$alpha = alpha_{global} * alpha_{pixel}$$
$$color_{out} = color_{in} * alpha + color_{target} * (1 - alpha)$$

Where alpha is a value between 0 (transparent) and 1 (opaque). If alpha blending is disabled the pixel is passed on unmodified. The alpha value can be interpolated over a triangle to create gradients. If this function is turned off (interpolation is disabled on triangle draws) then $alpha_{pixel}$ is set to 1.

2.6.8 Wishbone arbiter

Since two parts of the pipeline (fragment and blender) needs to access video memory, the arbiter makes certain only one of them can access the reader at once. The blender has the highest priority.

2.6.9 Wishbone master read

The wishbone reader handles all reads from video memory.

2.6.10 Renderer

The renderer calculates the memory address of the target pixel.

2.6.11 Wishbone master write

The wishbone master handles all writes to the video memory.

3 IO Ports

The Core has three wishbone interfaces:

- Wishbone slave – connects to the data bus of the OpenRISC processor. In the case of ORPSoC, this bus is connected through an arbiter. Supports standard wishbone communications, not any burst modes.
- Wishbone master read-only – connects to a video memory port with read access. Used for fetching textures and during blending.
- Wishbone master write-only – connects to a video memory port with write access. Used for rendering pixels to the framebuffer.

There is an interrupt enabled that can be connected to the interrupt pins on the or1200 CPU (in the supplied orpsoc_top it is connected to or1200_pic_ints[9]). For this interrupt to trigger, the correct bits in the control register has to be set.

4 Registers

Name	Addr	Width	Access	Description
CONTROL	0x00	32	RW	Control register
STATUS	0x04	32	R	Status register
ALPHA	0x08	32	RW	Global alpha register
COLORKEY	0x0c	32	RW	Colorkey register
TARGET_BASE	0x10	32	RW	Render target base
TARGET_SIZE_X	0x14	32	RW	Render target width
TARGET_SIZE_Y	0x18	32	RW	Render target height
TEX0_BASE	0x1c	32	RW	Texture 0 base
TEX0_SIZE_X	0x20	32	RW	Texture 0 width
TEX0_SIZE_Y	0x24	32	RW	Texture 0 height
SRC_P0_X	0x28	32	RW	Source pixel 0 x
SRC_P0_Y	0x2c	32	RW	Source pixel 0 y
SRC_P1_X	0x30	32	RW	Source pixel 1 x
SRC_P1_Y	0x34	32	RW	Source pixel 1 y
DEST_X	0x38	32	RW	Destination pixel x
DEST_Y	0x3c	32	RW	Destination pixel y
DEST_Z	0x40	32	RW	Destination pixel z
AA	0x44	32	RW	Transformation matrix coefficient
AB	0x48	32	RW	Transformation matrix coefficient
AC	0x4c	32	RW	Transformation matrix coefficient
TX	0x50	32	RW	Transformation matrix coefficient
BA	0x54	32	RW	Transformation matrix coefficient
BB	0x58	32	RW	Transformation matrix coefficient
BC	0x5c	32	RW	Transformation matrix coefficient
TY	0x60	32	RW	Transformation matrix coefficient
CA	0x64	32	RW	Transformation matrix coefficient
CB	0x68	32	RW	Transformation matrix coefficient
CC	0x6c	32	RW	Transformation matrix coefficient
TZ	0x70	32	RW	Transformation matrix coefficient
CLIP_P0_X	0x74	32	RW	Clip pixel 0 x
CLIP_P0_Y	0x78	32	RW	Clip pixel 0 y
CLIP_P1_X	0x7c	32	RW	Clip pixel 1 x
CLIP_P1_Y	0x80	32	RW	Clip pixel 0 y
COLOR0	0x84	32	RW	Color 0
COLOR1	0x88	32	RW	Color 1
COLOR2	0x8c	32	RW	Color 2
U0	0x90	32	RW	Texture coordinate 0
V0	0x94	32	RW	Texture coordinate 0
U1	0x98	32	RW	Texture coordinate 1
V1	0x9c	32	RW	Texture coordinate 1
U2	0xa0	32	RW	Texture coordinate 2
V2	0xa4	32	RW	Texture coordinate 2
ZBUFFER_BASE	0xa8	32	RW	Depth buffer base address

Each register is described in detail in the following sections, with information about what the purpose of each bit in the register is. The default value provided for each register is set when the device receives a reset signal.

4.1 Control Register (CONTROL)

Bit #	**Access**	**Description**
[31:20]	-	Reserved
[19]	W	Transform point
[18]	W	Forward point
[17:16]	RW	Active point
[15:14]	-	Reserved
[13]	W	Bézier inside shape
[12]	W	Interpolation
[11]	W	Curve write
[10]	W	Triangle write
[9]	W	Line write
[8]	W	Rect write
[7]	-	Reserved
[6]	RW	Z-buffer enable
[5]	RW	Clipping enable
[4]	RW	Colorkey enable
[3]	RW	Blending enable
[2]	RW	Texture0 enable
[1:0]	RW	Color depth

Default value: 0x00

Color depth is defined as follows:

Mode	**Color depth**
00	8 bit
01	16 bit
10	24 bit (not supported)
11	32 bit

The active point is defined as follows:

Mode	**Point id**
00	p0
01	p1
10	p2
11	p3

The operations **Forward point** and **Transform point** reads the current values of the active point and stores the x, y, z values in the correct register inside the device.

4.2 Status Register (STATUS)

Bit #	**Access**	**Description**
[31:16]	R	FIFO size
[15:1]	R	Reserved
[0]	R	Busy pin (high when busy)

Default value: –

4.3 Alpha (ALPHA)

Bit #	Access	Description
[31:24]	RW	Point 0 alpha
[23:16]	RW	Point 1 alpha
[15:8]	RW	Point 2 alpha
[7:0]	RW	Global alpha

Default value: 0xffffffff

The global alpha value is used in all rendering when alpha blending is enabled. **0xff** is full opacity, while **0x00** is full transparency (nothing rendered). When interpolation of triangles is activated, the point alpha values are used to find an interpolated alpha value for each pixel. This value is then multiplied with the global alpha before being used for blending.

4.4 Colorkey register (COLORKEY)

Bit #	Access	Description
[31:0]	RW	Colorkey

Default value: 0x00

By setting a colorkey certain pixels in a texture can be discarded in the fragment stage, providing a hard transparency. Depending on the color depth, a mask is applied to the color. Using 8 bit color, only the 8 least significant bits in the colorkey will be compared with the texture color during the check. The colorkey enable bit in the control register must be set to enable this functionality.

4.5 Target base address Register (TARGET_BASE)

Bit #	Access	Description
[31:2]	RW	Video Memory Address
[1:0]	-	Nothing

Default value: 0x00

4.6 Target size width Register (TARGET_SIZE_X)

Bit #	Access	Description
[31:0]	RW	Integer Width

Default value: 0x00

4.7 Target size y Register (TARGET_SIZE_Y)

Bit #	Access	Description
[31:0]	RW	Integer Height

Default value: 0x00

4.8 Texture 0 Base Register (TEX0_BASE)

Bit #	Access	Description
[31:2]	RW	Video Memory Address
[1:0]	-	Nothing

Default value: 0x00

4.9 Texture 0 size x Register (TEX0_SIZE_X)

Bit #	Access	Description
[31:0]	RW	Integer Width

Default value: 0x00

4.10 Texture 0 size y Register (TEX0_SIZE_Y)

Bit #	Access	Description
[31:0]	RW	Integer Height

Default value: 0x00

4.11 Source Pixel position 0 x Register (SRC_P0_X)

Bit #	Access	Description
[31:0]	RW	Integer x pos

Default value: 0x00

The source pixels are used to define a specific area in a texture to draw.

4.12 Source Pixel position 0 y Register (SRC_P0_Y)

Bit #	Access	Description
[31:0]	RW	Integer y pos

Default value: 0x00

4.13 Source Pixel position 1 Register (SRC_P1_X)

Bit #	Access	Description
[31:0]	RW	Integer x pos

Default value: 0x00

4.14 Source Pixel position 1 Register (SRC_P1_Y)

Bit #	Access	Description
[31:0]	RW	Integer y pos

Default value: 0x00

4.15 Destination Pixel position Register (DEST_X)

Bit #	Access	Description
[31:16]	RW	Signed Integer part
[15:0]	RW	Fractional part

Default value: 0x00

The control register flag active point decides the destination register inside the device. Points are pushed to the device by setting the forward or transform bit in the control register.

4.16 Destination Pixel position Register (DEST_Y)

Bit #	Access	Description
[31:16]	RW	Signed Integer part
[15:0]	RW	Fractional part

Default value: 0x00

4.17 Destination Pixel position Register (DEST_Z)

Bit #	Access	Description
[31:16]	RW	Signed Integer part
[15:0]	RW	Fractional part

Default value: 0x00

4.18 Matrix coefficient registers

The matrix coefficients are defined in the following way:

$$M = \begin{bmatrix} AA & AB & AC & TX \\ BA & BB & BC & TY \\ CA & CB & CC & TZ \end{bmatrix}$$

Each coefficient has a register, where the bits are defined as:

Bit #	Access	Description
[31:16]	RW	Signed Integer part
[15:0]	RW	Fractional part

The default matrix is set to no scaling, no rotation, no translation:

$$M_{default} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

4.19 Clip Pixel position 0 x Register (CLIP_P0_X)

Bit #	Access	Description
[31:0]	RW	Integer x

Default value: 0x00

4.20 Clip Pixel position 0 y Register (CLIP_P0_Y)

Bit #	Access	Description
[31:0]	RW	Integer y

Default value: 0x00

4.21 Clip Pixel position 1 x Register (CLIP_P1_X)

Bit #	Access	Description
[31:0]	RW	Integer x

Default value: 0x00

4.22 Clip Pixel position 1 y Register (CLIP_P1_Y)

Bit #	Access	Description
[31:0]	RW	Integer y

Default value: 0x00

4.23 Color Registers (COLOR0-2)

Bit #	Access	Description
[31:0]	RW	Color bits

Default value: 0x00

There are several color modes available (set in *video mode register*):

Mode	Format
32bpp	[31:24] is alpha channel. [23:16] is R, [15:8] is G and [7:0] is B
16bpp	[15:11] is R, [10:5] is B and [4:0] is G
8bpp gray	[7:0] sets both R, G and B values
8bpp palette	[7:0] sets the color index in the palette

5 Operation

All hardware accelerated operations draw pixels to the currently active surface (defined by TADR_REG and TSZE_REG). These operations are all affected by clip_p0 and clip_p1. No pixels that fall outside the clipping rectangle will be rendered.

5.1 Draw pixel

Input needed: dest_p0, color
Orgfx have no hardware-support for writing a single pixel to the video memory. However its possible to draw a line, rect or curve with the size of one pixel. The software API makes it possible to draw a pixel by writing directly to the memory (this is the most optimal way). Since the video memory can point to both the framebuffer and to textures, the same operation can be used to draw an arbitrary pixel to the screen and to load a texture into video memory.

5.2 Fill rect

Input needed: ctrl, dest_p0, dest_p1, color, [src_p0, src_p1]
Fill rect will fill the area of a rectangle created between the pixel dest_p0 and dest_p1 with color. If texturing is enabled, color will be taken from the active texture in the area between src_p0 and src_p1. This operation is hardware accelerated, and is activated by setting the Rect write bit in the control register.

5.3 Line

Input needed: dest_p0, dest_p1, color
Line will draw a line between the pixels dest_p0 and dest_p1 with color. This operation is hardware accelerated.

5.4 Fill triangle

Input needed: ctrl, dest_p0, dest_p1, dest_p2, color, [src_p0, src_p1]
Draw the pixels in the triangle created by dest_p0, dest_p1 and dest_p2.

5.5 Curve

Input needed: ctrl, dest_p0, dest_p1, dest_p2, dest_p3, color, [src_p0, src_p1]
Draws a cubic bézier curve. for a quadratic bézier curve, use the same value for dest_p1 and dest_p2.

5.6 Fill curve

Input needed: ctrl, dest_p0, dest_p1, dest_p2, dest_p3, color, [src_p0, src_p1]
Draws a filled cubic bézier curve. for a filled quadratic bézier curve, use the same value for dest_p1 and dest_p2.

6 Clocks

The wishbone slave uses the system wishbone bus clock at 50 Mhz, while the rest of the pipeline and the wishbone interfaces to the memory runs at 100Mhz.

7 Driver interface

The ORSoC graphics accelerator offers three different APIs to code against, two for bare metal when coding directly against the processor, and a Linux kernel module. The extended bare metal interface is a wrapper around the basic bare metal API, and makes coding easier by reducing the number of calls. The drawback is lesser control over the graphics card.

7.1 newlib

The basic library is provided in **orgfx.h** and **orgfx.c**.

The bare metal library declares a structure that can hold surfaces (both framebuffers and textures). Many functions take a pointer to one of these structures.

```
struct orgfx_surface
{
        unsigned int addr;
        unsigned int w;
        unsigned int h;
};
```

7.1.1 orgfx_init

Description: The orgfx_init must be called first to get other oc_gfx commands to work properly.

```
void orgfx_init(unsigned int memoryArea);
```

7.1.2 orgfx_vga_set_videomode

Description: Sets the video mode, width, height, bpp.

```
void orgfx_set_videomode(unsigned int width,
                         unsigned int height,
                         unsigned char bpp);
```

7.1.3 orgfx_vga_set_vbara

Description: Assign a memory address to ”Video Base Address Register A”.

```
void orgfx_vga_set_vbara(unsigned int addr);
```

7.1.4 orgfx_vga_set_vbarb

Description: Assign a memory address to ”Video Base Address Register B”.

```
void orgfx_vga_set_vbarb(unsigned int addr);
```

7.1.5 orgfx_vga_bank_switch

Description: Switches the framebuffer.

```
void orgfx_vga_bank_switch();
```

7.1.6 orgfx_init_surface

Description: Initialize a surface and return a control structure for it. This function increments an internal video memory stack pointer, so each surface will be allocated after the previous one in memory (starting at memoryArea set by orgfx_init). There is currently no memory management in place to recycle surface memory once it is no longer in use. The first surface initialized will point to the same memory that the video controller reads from, so it should be initialized with the width and height of the screen.

```
struct orgfx_surface
  orgfx_init_surface(unsigned int width,
                     unsigned int height);
```

7.1.7 orgfx_bind_rendertarget

Description: Binds a surface as the active render target. This function *must* be called before any drawing operations can be performed.

```
void orgfx_bind_rendertarget(struct orgfx_surface *surface);
```

7.1.8 orgfx_enable_cliprect

Description: Enables/disables clipping.

```
inline void orgfx_enable_cliprect(unsigned int enable);
```

7.1.9 orgfx_cliprect

Description: Sets the clipping rect. No pixels will be drawn outside of this rect (useful for restricting draws to a specific area of the render target). orgfx_bind_rendertarget will reset the clipping rect to the size of the surface.

```
inline void orgfx_cliprect(unsigned int x0,
                           unsigned int y0,
                           unsigned int x1,
                           unsigned int y1);
```

7.1.10 orgfx_srcrect

Description: Sets the source rectangle that will be used by texturing operations. This allows for only drawing a small part of a texture. orgfx_bind_tex0 will reset this to the size of the texture.

```
inline void orgfx_srcrect(unsigned int x0,
                          unsigned int y0,
                          unsigned int x1,
                          unsigned int y1);
```

7.1.11 orgfx_set_pixel

Description: Set a pixel on coordinate x,y to color. This is done in software by direct memory writes. This operation is not affected by the clipping rect!

```
inline void orgfx_set_pixel(unsigned int x,
                            unsigned int y,
                            unsigned int color);
```

7.1.12 orgfx_memcpy

Description: Copies memory from the processor to the video memory. Size is in 32-bit words. This function is intended to work with the output array of the sprite converter utility to load images into memory. Remember to bind a texture as the render target first!

```
void orgfx_memcpy(unsigned int mem[],
                  unsigned int size);
```

7.1.13 orgfx_set_color

Description: Sets the current drawing color.

```
inline void orgfx_set_color(unsigned int color);
```

7.1.14 orgfx_rect

Description: Draws a rect from (x0,y0) to (x1,y1) and fills it with the current drawing color. If texturing is enabled, the current texture will be drawn instead.

```
inline void orgfx_rect(unsigned int x0,
                       unsigned int y0,
                       unsigned int x1,
                       unsigned int y1);
```

7.1.15 orgfx_line

Description: Draws a line from (x0,y0) to (x1,y1) with the current drawing color. If texturing is enabled, the first pixel of the current texture will be drawn instead.

```
inline void orgfx_line(unsigned int x0,
                       unsigned int y0,
                       unsigned int x1,
                       unsigned int y1);
```

7.1.16 orgfx_triangle

Description: Draws a filled triangle of the space spanned by (x0,y0), (x1,y1) and (x2,y2). The order of the points is important, since triangles calculated to be counter clockwise will be discarded (backface culling). The interpolate flag indicates if flat coloring or interpolated coloring should be used. The interpolate flag *has* to be enabled for interpolated alpha, texture coordinates or depth is desired (flat coloring can be obtained by setting all three color registers to the same color).

```
inline void orgfx_triangle(unsigned int x0,
                           unsigned int y0,
                           unsigned int x1,
                           unsigned int y1,
                           unsigned int x2,
                           unsigned int y2,
                           unsigned int interpolate);
```

7.1.17 orgfx_triangle3d

Description: This function works the same way as the triangle function, but the Z-values are set.

```
inline void orgfx_triangle3d(unsigned int x0,
                             unsigned int y0,
                             unsigned int z0,
                             unsigned int x1,
                             unsigned int y1,
                             unsigned int z1,
                             unsigned int x2,
                             unsigned int y2,
                             unsigned int z2,
                             unsigned int interpolate);
```

7.1.18 orgfx_curve

Description: Draws a Quadratic curve between the points (x0,y0) and (x2,y2) with the control points (x1,y1). The three points form a triangle. The *inside* flag determines if the inside or outside of the curve is filled inside the triangle.

```
inline void orgfx_curve(unsigned int x0,
                        unsigned int y0,
                        unsigned int x1,
                        unsigned int y1,
                        unsigned int x2,
                        unsigned int y2,
                        unsigned int inside);
```

7.1.19 orgfx_enable_tex0

Description: Enables or disables texturing.

```
void orgfx_enable_tex0(unsigned int enable);
```

7.1.20 orgfx_bind_tex0

Description: Binds a surface as the current texture. Will reset the source rect.

```
void orgfx_bind_tex0(struct orgfx_surface* surface);
```

7.1.21 orgfx_enable_alpha

Description: Enables or disables alpha blending.

```
void orgfx_enable_alpha(unsigned int enable);
```

7.1.22 orgfx_set_alpha

Description: Sets the alpha blending value.

```
void orgfx_set_alpha(unsigned int alpha);
```

7.1.23 orgfx_enable_colorkey

Description: Enables or disables colorkey.

```
void orgfx_enable_colorkey(unsigned int enable);
```

7.1.24 orgfx_set_colorkey

Description: Sets the colorkey color.

```
void orgfx_set_colorkey(unsigned int colorkey);
```

7.2 Extended newlib

The extended library is provided in **orgfx_plus.h** and **orgfx_plus.c**, but **orgfx.c** also has to be compiled for it to work.

Instead of using surface structs directly, the extended API hides surface management by returning id tags for each surface. The screen surface (defined by id -1) is handled as a single surface, even when double buffering is enabled.

The driver defines the number of available surfaces (not counting the screen) with a static define. Change this if the default value is too low for your application.

7.2.1 orgfxplus_init

Description: Initializes the screen with the supplied video mode and returns an id for the screen.

```
int orgfxplus_init(unsigned int width,
                   unsigned int height,
                   unsigned char bpp,
                   unsigned char doubleBuffering);
```

7.2.2 orgfxplus_init_surface

Description: Unlike the basic API, this function both initializes a surface and loads a prepared image to it in one function call. The return value is an id that can be used to bind the surface. It changes render target during operation, but switches back to the last render target on completion. Since the screen(s) are already initialized by a call to init, they do not need to be loaded using this function.

```
int orgfxplus_init_surface(unsigned int width,
                           unsigned int height,
                           unsigned int mem[]);
```

7.2.3 orgfxplus_bind_rendertarget

Description: Binds a surface as the current render target.

```
void orgfxplus_bind_rendertarget(int surface);
```

7.2.4 orgfxplus_flip

Description: Swaps which buffer to draw on when using double buffering. Needs to be called once before anything shows up on screen!

```
void orgfxplus_flip();
```

7.2.5 orgfxplus_clip

Description: Sets the current clipping rect. This is reset to the size of the new render target when orgfxplus_bind_rendertarget is called.

```
inline void orgfxplus_clip(unsigned int x0,
                           unsigned int y0,
                           unsigned int x1,
                           unsigned int y1,
                           unsigned int enable);
```

7.2.6 orgfxplus_fill

Description: Draws a rectangle to the current render target with a flat color.

```
void orgfxplus_fill(unsigned int x0,
                    unsigned int y0,
                    unsigned int x1,
                    unsigned int y1,
                    unsigned int color);
```

7.2.7 orgfxplus_line

Description: Draws a line from (x0,y0) to (x1,y1) to the current render target with a flat color.

```
void orgfxplus_line(unsigned int x0,
                    unsigned int y0,
                    unsigned int x1,
                    unsigned int y1,
                    unsigned int color);
```

7.2.8 orgfxplus_triangle

Description: Draws a triangle between the points (x0,y0),(x1,y1) and (x2,y2) and fills it with a color.

```
void orgfxplus_triangle(unsigned int x0,
                        unsigned int y0,
                        unsigned int x1,
                        unsigned int y1,
                        unsigned int x2,
                        unsigned int y2,
                        unsigned int color);
```

7.2.9 orgfxplus_curve

Description: Draws a quadratic bÃ©zier curve from (x0,y0) to (x3,y3) with the control points (x1,y1) and (x2,y2).

```
void orgfxplus_curve(unsigned int x0,
                     unsigned int y0,
                     unsigned int x1,
                     unsigned int y1,
                     unsigned int x2,
                     unsigned int y2,
                     unsigned int x3,
                     unsigned int y3,
                     unsigned int color);
```

7.2.10 orgfxplus_draw_surface

Description: Draws a texture to the current render target.

```
void orgfxplus_draw_surface(unsigned int x0,
                            unsigned int y0,
                            unsigned int surface);
```

7.2.11 orgfxplus_draw_surface_section

Description: Draws a section of a texture defined by src0, src1 to the current render target.

```
void orgfxplus_draw_surface_section(unsigned int x0,
                                    unsigned int y0,
                                    unsigned int srcx0,
                                    unsigned int srcy0,
                                    unsigned int srcx1,
                                    unsigned int srcy1,
                                    unsigned int surface);
```

7.2.12 orgfxplus_colorkey

Description: Sets the colorkey color and enables or disables the use of the colorkey.

```
void orgfxplus_colorkey(unsigned int colorkey,
                        unsigned int enable);
```

7.2.13 orgfxplus_alpha

Description: Sets the alpha value and enables or disables the use of the alpha blending.

```
void orgfxplus_alpha(unsigned int alpha,
                     unsigned int enable);
```

7.3 Bitmap Fonts

7.3.1 orgfx_make_bitmap_font

Creates a orgfx_bitmap_font from a image. glyphSpacing space between two glyphs in the string and spaceWidth is the size of the space character.

```
orgfx_bitmap_font orgfx_make_bitmap_font(orgfx_tileset* glyphs,
                                          unsigned int glyphSpacing,
                                          unsigned int spaceWidth);
```

7.3.2 orgfx_put_text

Puts the text "str" on the screen with the specified "font" on position x0,y0.

```
void orgfx_put_text(orgfx_font* font,
                    unsigned int x0, unsigned int y0,
                    const char *str);
```

7.4 Vector Fonts

7.5 Linux

The current version of the core does not have a Linux driver.

7.6 Utilities

7.6.1 Sprite Maker

Since there is no libraries for loading images in the bare metal driver, a utility program is provided that converts an image into a format that can be loaded to the graphics accelerator. The Sprite Maker utility uses SDL and SDL_image to load images, and supports loading several basic formats, such as bmp, jpg, png, gif etc. The utility supports writing to 8-, 16-, 24- and 32-bits-per-pixel (must match the format set by orgfx_set_videomode). The width of the loaded image must be a multiple of 4 pixels (8 bpp), 2 pixels (16 bpp) or 1 bpp (24, 32 bpp) respectively.

The resulting output of the utility is a header file that can be included into your program. This header declares an array, which can be copied to memory and be used as a texture.

This is sample shows how the converter utility can be used:

```
./spritemaker image.png [bpp]
```

If bpp is not provided, the utility uses 8 bits-per-pixel. For an example of how to use the output of the converter, see section 8.

7.6.2 Mesh Maker

Similar to the image loading utility, there is a simple program that converts Maya obj files into a mesh format that is easy to load into the bare metal driver.

WRITE MORE

7.6.3 Fonter

The fonter is a application that converts a ttf file into a .h file that can be included in the project. if no input is given the application tries to open font.ttf and convert it to font.h.

```
./fonter [fontname.ttf] [output.h]
```

7.6.4 Regger

The regger is a application that keeps track of what register addresses is set in the RTL code and in the drivers.

8 Programming examples

The following piece of code shows how to use the extended interface for a bare metal implementation on the ORPSoCv2 platform. Bahamut_cc.png.h is a 186 by 248 pixel image with a pinkish background (rgb code ff00ff, or f81f in 16 bit). The header file is generated by the sprite maker utility at 16 bit color depth.

```
#include "orgfx_plus.h"

#include "Bahamut_cc.png.h"

int main(void)
{
    int i;

    // Initialize screen to 640x480-16@60
    // No double buffering
    int screen = orgfxplus_init(640, 480, 16, 0);

    // Initialize dragon sprite
    int bahamut_sprite =
        orgfxplus_init_surface(186, 248, Bahamut_cc);

    // Activate colorkeying
    orgfxplus_colorkey(0xf81f, 1);

    // Clear screen, white color
    orgfxplus_fill(0,0,640,480,0xffff);
    // Draw a few lines with different colors
    orgfxplus_line(200,100,10,10,0xf000);
    orgfxplus_line(200,100,351,31,0x0ff0);
    orgfxplus_line(200,100,121,231,0x00f0);
    orgfxplus_line(200,100,321,231,0xf00f);

    // Draw the dragon at different alpha settings
    orgfxplus_alpha(64,1);
    orgfxplus_draw_surface(100, 100, bahamut_sprite);
    orgfxplus_alpha(128,1);
    orgfxplus_draw_surface(120, 102, bahamut_sprite);
    orgfxplus_alpha(255,1);
    orgfxplus_draw_surface(140, 104, bahamut_sprite);

    while(1);
}
```

References