QUICK START GUIDE

Guidelines for Using the Graphics Library API

Ultra-Low Power MCU Family

A-MCUMSC-QSGA01EN v1.0

© 2020 Ambiq Micro, Inc.

Legal Information and Disclaimers

AMBIQ MICRO INTENDS FOR THE CONTENT CONTAINED IN THE DOCUMENT TO BE ACCURATE AND RELIABLE. THIS CONTENT MAY, HOWEVER, CONTAIN TECHNICAL INACCURACIES, TYPOGRAPHICAL ERRORS OR OTHER MISTAKES. AMBIQ MICRO MAY MAKE CORRECTIONS OR OTHER CHANGES TO THIS CONTENT AT ANY TIME. AMBIQ MICRO AND ITS SUPPLIERS RESERVE THE RIGHT TO MAKE CORRECTIONS, MODIFICATIONS, ENHANCEMENTS, IMPROVEMENTS AND OTHER CHANGES TO ITS PRODUCTS, PROGRAMS AND SERVICES AT ANY TIME OR TO DISCONTINUE ANY PRODUCTS, PROGRAMS, OR SERVICES WITHOUT NOTICE.

THE CONTENT IN THIS DOCUMENT IS PROVIDED "AS IS". AMBIQ MICRO AND ITS RESPECTIVE SUPPLIERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THIS CONTENT FOR ANY PURPOSE AND DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THIS CONTENT, INCLUDING BUT NOT LIMITED TO, ALL IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHT.

AMBIQ MICRO DOES NOT WARRANT OR REPRESENT THAT ANY LICENSE, EITHER EXPRESS OR IMPLIED, IS GRANTED UNDER ANY PATENT RIGHT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT OF AMBIQ MICRO COVERING OR RELATING TO THIS CONTENT OR ANY COMBINATION, MACHINE, OR PROCESS TO WHICH THIS CONTENT RELATE OR WITH WHICH THIS CONTENT MAY BE USED.

USE OF THE INFORMATION IN THIS DOCUMENT MAY REQUIRE A LICENSE FROM A THIRD PARTY UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF THAT THIRD PARTY, OR A LICENSE FROM AMBIQ MICRO UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF AMBIQ MICRO.

INFORMATION IN THIS DOCUMENT IS PROVIDED SOLELY TO ENABLE SYSTEM AND SOFTWARE IMPLEMENTERS TO USE AMBIQ MICRO PRODUCTS. THERE ARE NO EXPRESS OR IMPLIED COPYRIGHT LICENSES GRANTED HEREUNDER TO DESIGN OR FABRICATE ANY INTEGRATED CIRCUITS OR INTEGRATED CIRCUITS BASED ON THE INFORMATION IN THIS DOCUMENT. AMBIQ MICRO RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN. AMBIQ MICRO MAKES NO WARRANTY, REPRESENTATION OR GUARANTEE REGARDING THE SUITABILITY OF ITS PRODUCTS FOR ANY PARTICULAR PURPOSE, NOR DOES AMBIQ MICRO ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT, AND SPECIFICALLY DISCLAIMS ANY AND ALL LIABILITY, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL OR INCIDENTAL DAMAGES. “TYPICAL” PARAMETERS WHICH MAY BE PROVIDED IN AMBIQ MICRO DATA SHEETS AND/OR SPECIFICATIONS CAN AND DO VARY IN DIFFERENT APPLICATIONS AND ACTUAL PERFORMANCE MAY VARY OVER TIME. ALL OPERATING PARAMETERS, INCLUDING “TYPICALS” MUST BE VALIDATED FOR EACH CUSTOMER APPLICATION BY CUSTOMER’S TECHNICAL EXPERTS. AMBIQ MICRO DOES NOT CONVEY ANY LICENSE UNDER NEITHER ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS. AMBIQ MICRO PRODUCTS ARE NOT DESIGNED, INTENDED, OR AUTHORIZED FOR USE AS COMPONENTS IN SYSTEMS INTENDED FOR SURGICAL IMPLANT INTO THE BODY, OR OTHER APPLICATIONS INTENDED TO SUPPORT OR SUSTAIN LIFE, OR FOR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE AMBIQ MICRO PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR. SHOULD BUYER PURCHASE OR USE AMBIQ MICRO PRODUCTS FOR ANY SUCH UNINTENDED OR UNAUTHORIZED APPLICATION, BUYER SHALL INDEMNIFY AND HOLD AMBIQ MICRO AND ITS OFFICERS, EMPLOYEES, SUBSIDIARIES, AFFILIATES, AND DISTRIBUTORS HARMLESS AGAINST ALL CLAIMS, COSTS, DAMAGES, AND EXPENSES, AND REASONABLE ATTORNEY FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PERSONAL INJURY OR DEATH ASSOCIATED WITH SUCH UNINTENDED OR UNAUTHORIZED USE, EVEN IF SUCH CLAIM ALLEGES THAT AMBIQ MICRO WAS NEGLIGENT REGARDING THE DESIGN OR MANUFACTURE OF THE PART.

Revision History

Revision	Date	Description
1.0	April 10, 2020	Initial Release.

Reference Documents

Document ID	Description
A-MCUMSC-PGGA01EN	Ambiq MCU Graphics and GFX Library Programmer's Guide

Table of Contents

SECTION 1

Introduction

This Quick Start Guide for the GFX library provides simple and comprehensive guidelines on how to use the Library for developing purposes and includes the following topics:

- Command Lists
- Binding Textures
- Clipping
- Blending and Programming the Core
- Drawing

SECTION 2

Command Lists

A Command List (CL) is considered to be one of the most important features of the GPU. CL usage facilitates GPU and CPU decoupling, while its inherent re-usability greatly contributes to the decrease in computational effort of the CPU. This approach renders the overall architecture capable of drawing complicated scenes while keeping the CPU workload to the very minimum.

The design principles of CLs allow developers to extend the features of their application while optimizing its functionality at the same time. For instance, a CL is capable of jumping to another CL, thus forming a chain of seamlessly interconnected commands. In addition, a CL is able to branch to another CL and once the branch execution is concluded, resume its functionality after the branching point.

The GFX library helps developers to easily take advantage of all these features through certain basic function calls that trigger the whole spectrum of CL capabilities. A short presentation of the most fundamental subset of them is listed in the following sections.

2.1 Create

The most straightforward command for initiating a simple coding example is the **Create** command which is listed below.

```
nema_cmdlist_t nema_cl_create(void)
```

This fundamental command allocates and initializes a new Command List for later use.

2.2 Bind

This command sets the referred Command List as active. From that point on, each subsequent drawing call will incrementally be incorporated in the active Command List. At any time, all drawing operations should be called when there is a bound Command List.

```
nema_cl_bind_cmdlist(nema_cmdlist_t * cl)
```

2.3 Unbind

Unbind the currently bound Command List.

```
nema_cl_unbind_cmdlist(void)
```

2.4 Submit

Submit the referred Command List for execution. If this CL is currently the one that is bound, this call unbinds it. When a CL is submitted for execution, it should never be altered until it finishes execution. Writing in such a CL results in undefined behavior.

```
nema_cl_submit_cmdlist(nema_cmdlist_t * cl)
```

A typical routine for drawing would be the following:

```
nema_cmdlist_t cl = nema_cl_create();  //Create a new CL
nema_cl_bind_cmdlist(&cl);             //Bind it

/* Drawing Operations */               // Draw scene

nema_cl_unbind_cmdlist();              //Unbind CL (optionally)
nema_cl_submit_cmdlist(&cl);           //Submit CL for execution
```

SECTION 3

Binding Textures

Every drawing operation should have an effect on a given destination texture. The texture must reside in some memory space which is visible to the GPU.

```
void nema_bind_dst_tex(uint32_t baseaddr_phys,
                       uint32_t width, uint32_t height,
                       nema_tex_format_t format, int32_t stride)
```

The above function binds a texture to serve as destination. The texture's attributes (GPU address, width, height, format and stride) are written inside the bound CL. Each subsequent drawing operation will have an effect on this destination texture.

Most common graphics operations include some kind of image blitting (copying), like drawing a background image, GUI icons or even font rendering. The following command binds a texture to be used as foreground:

```
void nema_bind_src_tex(uint32_t baseaddr_phys,
                       uint32_t width, uint32_t height,
                       nema_tex_format_t format, int32_t stride,
                           nema_tex_mode_t mode);
```

This function call has a very similar functionality to the NemaGFX_bind_dst_tex. It has one extra argument, NEMA_tex_mode_t mode, that determines how to read a texture (point/bilinear sampling, wrapping mode etc).

The above example can now be extended as follows:

```
nema_cmdlist_t cl = nema_cl_create();      // Create a new CL
nema_cl_bind_cmdlist(&cl):                 // Bind it

                                  // Bind Destination Texture:
nema_bind_dst_tex(DST_IMAGE,      // Destination address
             320, 240,            // width, height
             NEMA_RGBA8888,       // Image format (32bit, rgba)
             320*4);              // Stride in bytes (width*4 bytes per pixel)

                                  // Bind Foreground Texture:
nema_bind_src_tex(SRC_IMAGE,      // Source address
             320, 240,            // width, height
             NEMA_RGBA8888,       // Image format (32bit, rgba)
             320*4                // Stride in bytes (width*4 bytes per pixel)
             NEMA_FILTER_PS);     // Do point sampling (default option)

/* Drawing Operations */          // Draw scene

nema_cl_unbind_cmdlist();         // Unbind CL (optionally)
nema_cl_submit_cmdlist(&cl);      // Submit CL for execution
```

SECTION 4

Clipping

When drawing a scene, it is often necessary to be able to define a rectangular area that the GPU is allowed to draw. This way, if some parts of a primitive (e.g., a triangle) falls outside the clipping area, that part is not going to be drawn at all, assuring correctness, better performance and improved power efficiency. The Clipping Rectangle can be defined as follows:

```
void nema_set_clip(int32_t x, int32_t y, int32_t w, int32_t h)
```

This function defines a Clipping Rectangle whose upper left vertex coordinates are (x, y) and its dimensions are w·h.

The default Clipping Rectangle usually is the entire canvas. In the above examples, we used textures with dimensions of 320x240. So, adding Clipping would result the following:

```
nema_cmdlist_t cl = nema_cl_create();      // Create a new CL
nema_cl_bind_cmdlist(&cl);                 // Bind it

                                   //Bind Destination Texture:
nema_bind_dst_tex(DST_IMAGE,       //Destination address
            320, 240,              // width, height
            NEMA_RGBA8888,         // Image format (32bit, rgba)
            320*4);                // Stride in bytes (width*4 bytes per pixel)

                                   // Bind Foreground Texture:
nema_bind_src_tex (SRC_IMAGE,      // Source address
            320, 240,              // width, height
            NEMA_RGBA8888,         // Image format (32bit, rgba)
            320*4                  // Stride in bytes (width*4 bytes per pixel)
            NEMA_FILTER_PS);       // Do point sampling (default option)

nema_set_clip(0, 0, 320, 240);     // Define a 320x240 Clipping Rectangle

/* Drawing Operation */            // Draw scene

nema_cl_unbind_cmdlist();          // Unbind CL (optionally)
nema_cl_submit_cmdlist(&cl);       // Submit CL for execution
```

SECTION 5

Blending and Programming the Core

When building a graphical interface, the developer has to define what would be the result of drawing a pixel on the canvas. Since the canvas already contains the previous drawn scene, there must be a consistent way to determine how the source or foreground color (the one that is going to be drawn) will blend with the destination or background color that is already drawn. The source pixel can be fully opaque, thus will be drawn over the destination one, or it can be translucent and the result would be a blend of both the source and destination colors.

For example, blitting a background image of a GUI would require the Source Texture to cover entirely whatever is already drawn on the canvas. Afterwards, blitting an icon would require the background to be partially visible on the translucent areas of the icon. In order to make this possible, the GFX library incorporates a powerful set of predefined blending modes that allow the developer to build functional and eye catching applications:

```
void nema_set_blend_fill(nema_blend_mode_t blending_mode)
void nema_set_blend_blit(nema_blend_mode_t blending_mode)
```

These two functions refer to blending when filling a primitive (e.g., triangle) with a color or when blitting a texture respectively.

The previous example, after setting the correct blending mode for blitting a background texture, would evolve to the following:

```
nema_cmdlist_t cl = nema_cl_create();   // Create a new CL
nema_cl_bind_cmdlist(&cl);              // Bind it

                                  // Bind Destination Textures:
nema_bind_dst_tex(DST_IMAGE,      // Destination address
        320, 240,                 // width, height
        NEMA_RGBA8888,            // Image format (32bit, rgba)
        320*4);                   // Stride in bytes (width*4 bytes per pixel)

                                  // Bind Foreground Texture:
nema_bind_src_tex (SRC_IMAGE,     // Source address
         320, 240,                // width, height
         NEMA_RGBA8888,           // Image format (32bit, rgba)
         320*4                    // Stride in bytes (width*4 bytes per pixel)
         NEMA_FILTER_PS);         // Do point sampling (default option)

nema_set_clip(0, 0, 320, 240);    // Define a 320x240 Clipping Rectangle

nema_set_blend_blit(NEMA_BL_SRC); // Program the Core to draw the source color
                                  // without blending it with the destination
                                  // texture

/* Drawing Operations */          // Draw scene

nema_cl_unbind_cmdlist();         // Unbind CL (optionally)
nema_cl_submit_cmdlist(&cl);      // Submit CL for execution
```

SECTION 6

Drawing

Finally, after setting up the above, the CL contains all the information needed to blit an image or fill a Geometric Primitive with color. The GFX library has a rich set of functions to do that. For the example above, let's assume that we need to draw a background 320x240 image, starting at screen coordinate (0,0) (the upper left corner of the canvas), and then draw a red rectangle that starts at point (20, 30) with dimensions 100x200:

```
nema_cmdlist_t cl = nema_cl_create();   // Create a new CL
nema_cl_bind_cmdlist(&cl);              // Bind it

                                  // Bind Destination Texture:
nema_bind_dst_tex(DST_IMAGE,      // Destination address
              320, 240,           // width, height
              NEMA_RGBA8888,      // Image format (32bit, rgba)
              320*4);             // Stride in bytes (width*4 bytes per pixel)

                                  // Bind Foreground Texture:
nema_bind_src_tex (SRC_IMAGE,     // Source address
              320, 240,           // width, height
              NEMA_RGBA8888,      // Image format (32bit, rgba)
              320*4,              // Stride in bytes (width*4 bytes per pixel)
              NEMA_FILTER_PS);    // Do point sampling (default option)

nema_set_clip(0, 0, 320, 240);    // Define a 320x240 Clipping Rectangle

nema_set_blend_blit(NEMA_BL_SRC); // Program the Core to draw the source
                                  // texture without blending it with the
                                  // destination texture
nema_blit(0, 0);                  // Blit the bound Source Texture to
                                  // Destination Texture

nema_set_blend_fill(DEMA_BL_SRC); // Program the Core to fill the Geometric
                                  // Primitive without blending it with the
                                  // destination texture

nema_fill_rect(20, 30, 100, 200, RED);// Fill a rectangular area with red color

nema_cl_unbind_cmdlist();         // Unbind CL (optionally)
nema_cl_submit_cmdlist(&cl);      // Submit CL for execution
```

The overall process described in the previous paragraphs, produces the output presented in Figure 6-1.

Figure 6-1: Drawing Output

Original Empty Frame Buffer Background Final Output

SECTION 7

Contact Information

Table 7-1: Contact Information

Address:	Ambiq Micro, Inc 6500 River Place Boulevard, Building 7, Suite 200 Austin, TX 78730-1156
Phone:	+1(512) 879-2850
Website	ambiqmicro.com
Emails:	info@ambiqmicro.com sales@ambiqmicro.com
Technical Support:	support.ambiqmicro.com

© 2020 Ambiq Micro, Inc. All rights reserved.

6500 River Place Boulevard, Building 7, Suite 200, Austin, TX 78730
www.ambiqmicro.com
sales_americas@ambiqmicro.com
+1 (512) 879-2850

A-MCUMSC-QSGA01EN v1.0
April 2020