ALP Application Note: Grayscale using Bit-plane Lookup Table

1 Table of Contents

2 Introduction

Technical applications of Texas Instruments DLP® chips (also known as DMD, Digital Micromirror Device) often need to illuminate an object with gray scale patterns. Lithography processes, for example, use structured light to apply an image to a photo sensitive substrate.

Having different levels of light intensity enables edge smoothing and resolution enhancement algorithms (see Figure 1 and Figure 2). This application note shows possibilities of the ViALUX ALP API (application programming interface) to generate grayscale images on V-Module DMDs.

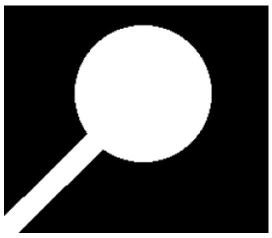


Figure 1: Binary pattern (black and white pixels)

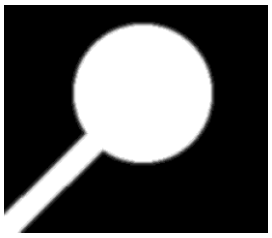


Figure 2: Gray scale pattern (smooth edges)

Starting with the standard grayscale mode, it will show more elaborated approaches using bit-plane lookup tables. Finally, implementation details are demonstrated using the sample application “ALP BPLUT Sample”. This sample application shows these concepts both in source code and executable on a V-Module. For programming details of particular ALP functions, refer to the ALP-4.3 API description as usual.

3 Grayscale Generation with a DMD

DMDs are spatial light modulators with binary behavior. They consist of pixels (micro mirrors) that have two active states. The optical projection system converts these states into dark and bright pixels on the substrate, screen, or generally the object to be illuminated. Intermediate brightness levels (gray) are not possible for different pixels at a single point in time (see Figure 3).

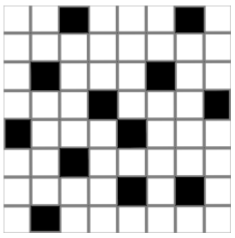
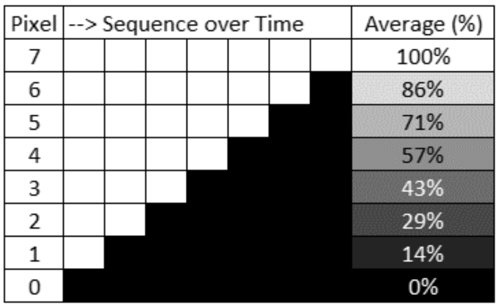


Figure 3: Binary DMD display

Processes that need gray-scale usually only require light doses per pixel meaning the level of light integrated over time. In other words, what counts is the average power, not the peak power. Figure 4 illustrates this for eight different light levels (0 to 7).



Pixel	--> Sequence over Time							Average (%)
7	⬜	⬜	⬜	⬜	⬜	⬜	⬜	100%
6	⬜	⬜	⬜	⬜	⬜	⬜	⬛	86%
5	⬜	⬜	⬜	⬜	⬜	⬛	⬛	71%
4	⬜	⬜	⬜	⬜	⬛	⬛	⬛	57%
3	⬜	⬜	⬜	⬛	⬛	⬛	⬛	43%
2	⬜	⬜	⬛	⬛	⬛	⬛	⬛	29%
1	⬜	⬛	⬛	⬛	⬛	⬛	⬛	14%
0	⬛	⬛	⬛	⬛	⬛	⬛	⬛	0%

Figure 4: Generate average pixel levels by sequential display of binary frames over time

3.1 ALP bit plane processing

ALP takes advantage of the standard code of numbers in digital systems. Integer values are composed of bits, e.g. a byte is an eight-bit number. Each bit contributes to the value by the power of two of its position (1, 2, 4, 8…). This is called the weight of a bit.

ALP extracts one binary image from the gray scale image for each bit position. These binary images are called bit planes. Each bit plane needs to be displayed for a total duration according to its weight.

The bit plane with the smallest weight is the least-significant bit (LSB), the highest weight is the most-significant bit (MSB).

3.2 Standard bit plane timing

The standard display mode of ALP avoids switching the DMD and keeps each bit plane stable for its total duration. The resulting sequence (Figure 5) differs from Figure 4 above, but still results in the same average per pixel value.

Pixel	--> Sequence over Time			Average (%)	Note
7	⬜	⬜	⬜	100%	
6	⬜	⬜	⬛	86%	
5	⬜	⬛	⬜	71%	
4	⬜	⬛	⬛	57%	
3	⬛	⬜	⬜	43%	
2	⬛	⬜	⬛	29%	
1	⬛	⬛	⬜	14%	
0	⬛	⬛	⬛	0%	
	4	2	1		Weight 2^{BPL}
	2	1	0		Bitplane BPL

Figure 5: ALP Standard mode: Assign bit planes BPL of the image to intervals of different weights

ALP provides the ALP_BITNUM setting to allow the end-user to select the trade-off between repetition rate (Maximum Frames per Second = 1/*MinPictureTime*) and available gray scale steps (see Table 1).

ALP_BITNUM	Grayscale levels	Maximum Frames per Second
1	2 (0 to 1)	22727
2	4 (0 to 3)	9174
3	8 (0 to 7)	5555
4	16 (0 to 15)	3496
5	32 (0 to 31)	2016
6	64 (0 to 63)	1091
7	128 (0 to 127)	569
8	256 (0 to 255)	290
9	512 (0 to 511)	146
10	1024 (0 to 1023)	73
11	2048 (0 to 2047)	37
12	4096 (0 to 4095)	18

Table 1: Trade off between gray-scale resolution and maximum frame rate (e.g. DMD type DLP7000)

The result is a perfectly linear generation of light dose per pixel. The draw-back of this standard approach are long duration of higher bit planes, and a long picture time in general.

4 Customize Sequence using Bit-plane Lookup Tables

Some applications require higher switching frequency. One example is the illumination of a continuously moving substrate.

ALP supports this setup by means of the scrolling extension, see also the ALP API description (*AlpSeqControl*, ALP_LINE_INC). Movement causes blurring effects that can only be restricted by small scrolling steps (ALP_LINE_INC=1). High through-put can then only be realized by high frame rates, that is in binary display (ALP_BITNUM=1).

The following sections show, how the bit-plane lookup table feature (ALP_BITPLANE_LUT_MODE) helps to get integrated gray-scale doses on a substrate pixel even with binary display.

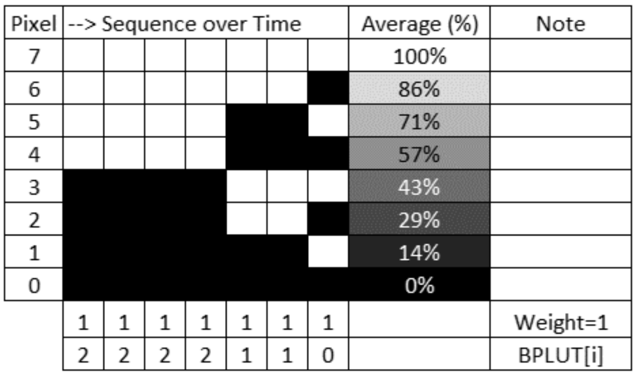
4.1 Basic concept of ALP_BITPLANE_LUT_MODE

The approach keeps using bit planes extracted from gray scale images as usual by ALP. Sequences are allocated just like normal gray-scale sequences (*AlpSeqAlloc*, e.g. *BitPlanes*=3 for eight gray values 0 to 7).

The binary sequence (ALP_BITNUM=1) is displayed using equal duration for each frame (*PictureTime*). Each frame will cause one scrolling step, defining the substrate movement speed.

So far, this is the standard ALP behavior. The DMD still always displays the MSB. The ALP_BITPLANE_LUT_MODE extension now adds the possibility to select different bit planes. After switching to ALP_BITPLANE_LUT_MODE=ALP_BITPLANE_LUT_FRAME (*AlpSeqControl*) the user can define the sequence of bit planes.

Figure 6 for example repeats a similar display like Figure 5. It illustrates that all bit planes have the same weight, but now are displayed a different number of times.



Pixel	--> Sequence over Time							Average (%)	Note
7	□	□	□	□	□	□	□	100%	
6	□	□	□	□	□	□	■	86%	
5	□	□	□	□	■	■	□	71%	
4	□	□	□	□	■	■	■	57%	
3	■	■	■	■	□	□	□	43%	
2	■	■	■	■	□	□	■	29%	
1	■	■	■	■	■	■	□	14%	
0	■	■	■	■	■	■	■	0%	
	1	1	1	1	1	1	1		Weight=1
	2	2	2	2	1	1	0		BPLUT[i]

Figure 6: Emulate ALP Standard grayscale generation using ALP_BITPLANE_LUT_MODE=ALP_BITPLANE_LUT_FRAME

This is the starting point of distributing the bit planes over time in another, more high-frequency way.

The highly significant bit planes can be distributed in the sequence to be displayed. Figure 7 shows how one could generate such a regular pattern. The initially empty sequence is filled by placing the MSB into each other index. The next bit plane is distributed the same way, taking only empty fields into account. Continue this until you reach the LSB.

--> Sequence over Time							Average (%)	Note
2		2		2		2		4*Bit 2 (MSB)
2	1	2		2	1	2		2*Bit1
2	1	2	0	2	1	2		1*Bit0 (LSB)

Figure 7: Generate a high-frequency display sequence by distribution of heavy-weight bit planes. Example: 3 bit planes.

In order to display bit-planes in this sequence, prepare a structure of type *tBplutWrite* and sent to the device using *AlpProjControlEx*.

Note: In the ALP bit-plane lookup table, the numbers are inverted, meaning that MSB is always bit 0. The C++ implementation of this algorithm can be found in section 5.2.

The resulting display generates the eight possible pixel values as illustrated in Figure 8.

Pixel	--> Sequence over Time							Average (%)	Note
7	□	□	□	□	□	□	□	100%	
6	□	□	□	■	□	□	□	86%	
5	□	■	□	□	□	■	□	71%	
4	□	■	□	■	□	■	□	57%	
3	■	□	■	□	■	□	■	43%	
2	■	□	■	■	■	□	■	29%	
1	■	■	■	□	■	■	■	14%	
0	■	■	■	■	■	■	■	0%	
	1	1	1	1	1	1	1		Weight=1
	2	1	2	0	2	1	2		BPLUT[i]

Figure 8: Realize high-frequency display sequence of 3 bit planes.

4.2 Gray scale generation in scrolling mode

Until this point, all the consideration has focused on DMD pixel. We have seen how each DMD pixel can produce gray-scale light doses over time. In non-moving setup, the same applies to substrate pixels, because each micro-mirror is imaged to a constant point on the substrate.

But lithographic processes usually have a moving setup. The substrate or the DMD moves in a linear path, exposing a stripe. This is then repeated in order to process the full area stripe by stripe.

As said before, linear movement of the substrate is supported by the ALP scrolling extension (ALP_LINE_INC). Now, the same substrate pixels are imaged by different DMD pixels over time. This allows different approaches to expose the light dose.

4.3 One bit plane per frame (ALP_BITPLANE_LUT_FRAME)

Figure 9 below illustrates the movement of the substrate over the vertical axis. The ALP sequence scrolls accordingly. The binary DMD images at each point in time are next to each other, over the horizontal axis.

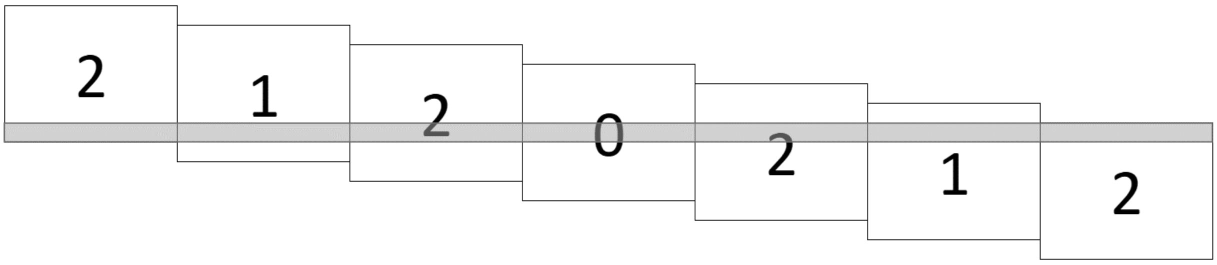


Figure 9: ALP_BITPLANE_LUT_FRAME: display one complete display at a time. The high-lighted substrate stripe collects a light dose of 4*Bit2 + 2*Bit1 + 1*Bit0.

An area of the substrate receives its correct gray-scale light dose of three bit planes that are consecutively displayed. At each point in time, one complete bit plane is shown. This sequence can be repeated arbitrary numbers of times to expose a stripe of virtually infinite length.

One constraint, however, is that the DMD area should completely pass after an integer multiple 7*n of scrolling steps (i.e. DMD rows). More generally, the active DMD size should be a multiple of $2^{\text{Bitplanes}}$-1 rows. ALP helps to realize this by reducing the area-of-interest by some rows, using ALP_SEQ_DMD_LINES.

4.4 Combine bit planes (ALP_BITPLANE_LUT_ROW)

Another approach generates binary frames from all available bit planes. ALP allows to divide the DMD in regions and select the bit planes to display in every region. This selection remains steady while the sequence scrolls.

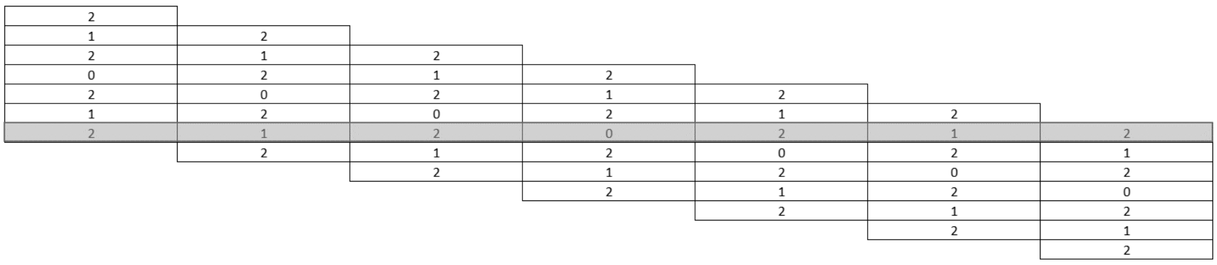


Figure 10: ALP_BITPLANE_LUT_ROW: display one complete display at a time. The high-lighted substrate stripe collects a light dose of 4*Bit2 + 2*Bit1 + 1*Bit0.

The same restriction regarding display height applies. It is even more obvious here that active DMD rows must match an integer multiple of $2^{\text{Bitplanes}}$-1.

5 Implementation Details and Application “ALP BPLUT Sample”

The ALP BPLUT Sample application is a console program. The interactive user interface consists of text output (see Figure 11) and keyboard input (see Figure 12). A pointer devices (such as a mouse) is not required. Some demos are only short. They pass very quickly, but the user always has the option to simple repeat them by hitting the space bar. This is possible while keeping the eyes on the DMD. One complete walk-through is executed when always pressing the enter key (also known as return key).

Of course, there must be some content to be displayed. The ALP BPLUT Sample reads this content from an image file. This allows to compare observed and expected display. The user can specify the file name as a command line parameter, or, for convenience, by drag and drop an image file to the executable program (ALP BPLUT Sample.exe). The fall-back default file name is “SampleImage.png”. The image should be a long stripe (significantly higher than wide), because it will be displayed in scrolling mode.

```
ALP BPLUT Sample
(c) 2023 ViALUX GmbH
Using ALP #13004 (1920*1080 pixels)
Bitplanes to be tested: 4 (gray values 0..15)
Load image from file: …\ALP BPLUT Sample\stripe.png
scaled image to ALP sequence of 1920*7500 pixels.
```

Figure 11: Welcome message and successful initialization

```
Select demo: Space=repeat, Return=next, 1..9=jump, f=toggle frame rate, q=quit
-> next (demo )
```

Figure 12: Prompt for keyboard input of what to do next

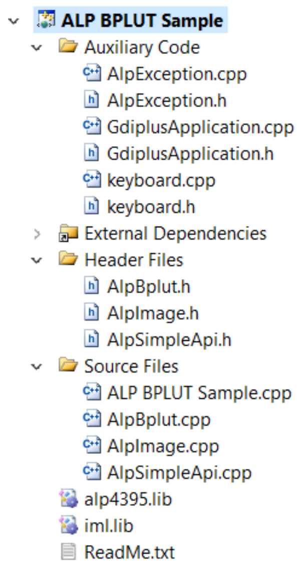
Several gray-scale demo cases are implemented. There is always a description of what to expect to see on the DMD for a particular demo case (e.g. Figure 13).

```
Standard ALP grayscale display:
What to expect?
- step through the sequence one DMD per frame
  that means 9 frames per sequence
- use standard grayscale algorithm with 4 bit-planes
- use default display timing: 33ms per frame
- sequence is repeated
Running. Hit a key to stop...
Standard ALP grayscale display done.
```

Figure 13: Explanation of a demo case

5.1 Program structure

One primary goal of ALP sample code is to clearly point out the ALP API function calls that are required to realize a use case. In order to do so, some “Auxiliary code” has been moved away from the application code. Figure 14 shows a list of all files contained in the Visual Studio project.



```
ALP BPLUT Sample
  Auxiliary Code
    AlpException.cpp
    AlpException.h
    GdiplusApplication.cpp
    GdiplusApplication.h
    keyboard.cpp
    keyboard.h
  External Dependencies
  Header Files
    AlpBplut.h
    AlpImage.h
    AlpSimpleApi.h
  Source Files
    ALP BPLUT Sample.cpp
    AlpBplut.cpp
    AlpImage.cpp
    AlpSimpleApi.cpp
  alp4395.lib
  iml.lib
  ReadMe.txt
```

Figure 14: ALP BPLUT Sample source code project structure

5.1.1 Class CAlpSimpleApi

ALP API function calls are hidden inside the class CAlpSimpleApi. The demo case demoGrayScale() for instance just calls alp.projStartCont(), see Figure 15. The ALP device ID, sequence ID, and error handling is completely done inside (Figure 16).

At the time of writing, CAlpSimpleApi only supports one ALP sequence. The supported set of ALP functions is incomplete. But it can serve as a starting point, extendable as required.

```
…
        waitForKeyToStart();
        alp.projStartCont();
        pause("Running. Hit a key to stop...");
        alp.projStop();
                                         …
```

Figure 15: Code to start display (demo case demoGrayScale())

```
CAlpSimpleApi& CAlpSimpleApi::projStartCont() throw (...)
{
        AlpError(AlpProjStartCont(mDevId, mSeqId), "Start continuous display");
        return *this;
}
```

Figure 16: Implementation detail of CAlpSimpleApi::projStartCont()

5.1.2 Error handling with class CAlpException

The approach is to abort the program in case of any error. This is implemented in the function AlpError(long apiReturn, string operation). If an ALP function call succeeds, then it returns ALP_OK. In this case, AlpError() just continues. In all other cases, it throws an exception of type CAlpException, providing the text about the operation that caused the error.

The main() function just catches these exceptions, not requiring any further control flow (like if-then-else branching) on errors.

5.1.3 Class CAlpImage and DMD Type support

The class CAlpImage manages a memory buffer for the image data in a format suitable for *AlpSeqPut*(). Image files are read using another ViALUX library, iml.dll.

CAlpImage supports all DMD types. It scales the image file to the DMD width, preserving the aspect ratio. Note that this approach is only suitable for demonstration. Final lithography applications should always supply pixel-accurate image data.

CAlpImage takes internally care to allocate a memory buffer large enough to hold an integer number of ALP pictures. This is required by *AlpSeqPut*. And it additionally implements the size calculations to add black areas (“fence”) before and after the payload data, see also section 5.5.1.

5.1.4 File ALP BPLUT Sample.cpp and Demo cases

This is the main demo file. It contains implementations of the following functions:

- *main*() does one-time ALP initialization, handles image files, and implements the user interface for navigating through demo cases.
- *demoGrayScale*(): ALP standard gray-scale generation; step frame by frame (no scrolling); default timing (30 fps)
- *demoGrayScaleScrolling*(): ALP standard gray-scale generation;
  scroll line by line; maximum frame rate, available for gray-scale
- *demoBinaryScaleScrolling*(): reduce number of bit-planes to 1 (binary); no BPLUT (i.e. display only MSB); *selectable pictureTime (to show possible speed-up)*
- *demoBinaryBplutFrame*(): *use BPLUT = FRAME (show all bit-planes, one at a time);* selectable pictureTime
- *demoBinaryBplutRow*(): *use BPLUT = ROW (combine all bit-planes to DMD frames);* selectable pictureTime

5.2 Algorithm for high frequency bit-plane lookup table: CAlpBplut::generateDeltaSigma()

The class CAlpBplut manages a memory buffer for the ALP data structure *tBplutWrite* and implements functions to generate particular patterns in this LUT.

Member function *generateDeltaSigma*(), as shown in Figure 17, realizes the algorithm described by Figure 7, section 4.1.

It starts by filling the MSB (ALP bit-plane 0) in each second entry (initial stepWidth=2), then continuing with bit-plane 1 in each second gap (stepWidth=4) and so on. This way, it can generate a sequence of length entryCount=$2^{\text{Bitplanes}}$-1. Each bit-plane occurs 2^{bitplane} times.

```
// Write bit-planes to the memory in a high-frequency pattern,
// beginning with the most-significant (bit plane 0 in ALP).
unsigned char bitplane=0;
long firstIndex=0;
long stepWidth=2;        //< The stepWidth increases while bit weight decreases.

while (firstIndex<entryCount)
{
        // Skip throught the LUT by stepWidth, processing all entries for the
        // current bitplane:
        for (long i=firstIndex; i<entryCount; i+=stepWidth)
        {
                // write a block of same bit-planes
                mBplut->BitPlanes[i] = bitplane;
        }

        // Next bitplane entries start right in the middle of the first "gap",
        // and have the double stepWidth.
        firstIndex += stepWidth/2;
        stepWidth *= 2;
        bitplane++;
}
```

Figure 17: Algorithm generateDeltaSigma() bit-plane lookup table

5.3 demoBinaryBplutFrame()

The BPLUT_FRAME demo case displays the image in scrolling mode, using a binary frames, but switches the bit-planes to still generate correct gray-scale pixels.

It uses the high-frequency bit-plane lookup table (Figure 18). The source code comments provide hints to modify the bit-plane lookup table to lower the switching frequency, but still generating valid gray-scale pixels.

```
// Generate a high-frequency bit-plane lookup table:
CAlpBplut bplutData;
bplutData.generateDeltaSigma(alp.getSeqBitplanes());
// Hint: also try out e.g. bplutData.generateAlpStandard

// If wanted, the bitplane switch frequency can be reduced:
// bplutData.distribute( alp.getBinaryAoiHeight()/bplutData.getSize() );
```

Figure 18: BPLUT generation in demoBinaryBplutFrame()

The sequence setup for this mode is shown in Figure 19. Some of these settings need a call to *AlpSeqTiming()* to be activated (Figure 20).

Note that in contrast to the ROW mode, the length of bit-plane lookup table does not correlate to the DMD size and must be adjusted properly (ALP_BITPLANE_LUT_ENTRIES).

```
// Use binary display, AOI, BPLUT; the required seqTiming() call is below
alp.seqControl(ALP_BITNUM, 1).seqControl(ALP_BIN_MODE, ALP_BIN_UNINTERRUPTED)
        .seqControl(ALP_SEQ_DMD_LINES, alp.getBinaryAOI())
        .seqControl(ALP_BITPLANE_LUT_MODE, ALP_BITPLANE_LUT_FRAME)
        .seqControl(ALP_BITPLANE_LUT_ENTRIES, bplutData.getSize())
        .setBplut(bplutData)
        .setSeqScroll(1, 0, scrollToRow);
```

Figure 19: Sequence setup for demoBinaryBplutFrame()

```
setTiming(alp, pictureTime, scrollToRow+1);       // calls alp.seqTiming()
```

Figure 20: AlpSeqTiming activates the bit-plane mode settings

The demo case reports the first entries of the bit-plane lookup table, and the expected run time of the display (depending on DMD type, image size, frequency selection, e.g. Figure 21).

```
Scrolling with bit-plane lookup per DMD FRAME (binary uninterrupted display):
What to expect?
- BPLUT causes bit-planes to be displayed in the following order:
  0 1 0 2 0 1 0 3 0 1 0 2 0 1 0
- use reduced display speed: 2805ms per sequence
- sequence is started for one pass only
Scrolling with bit-plane lookup per DMD FRAME, done.
```

Figure 21: Console output of demo case demoBinaryBplutFrame()

5.4 demoBinaryBplutRow()

The BPLUT_ROW demo is similar to the BPLUT_FRAME demo case. It displays the image in scrolling mode, using a binary frames. Even the bit-plane lookup table is the same.

But it does not switches bit-planes frame by frame. Instead it combines them row by row to binary DMD frames, as explained in section 4.4.

At first glance, the DMD seems to show the same gray-scale sequence. But a closer look reveals the fix assignment of bit-planes to DMD rows.

Figure 22 shows the sequence setup for mode ALP_BITPLANE_LUT_ROW.

Note that a BPLUT entry is required for every active DMD row. The code realizes this by calling the CAlpBplut member function *bplutData.duplicate()*. This function copies the generated bit-planes multiple times, until the requested length is reached.

```
// Use binary display, AOI, BPLUT; the required seqTiming() call is below
alp.seqControl(ALP_BITNUM, 1).seqControl(ALP_BIN_MODE, ALP_BIN_UNINTERRUPTED)
        .seqControl(ALP_SEQ_DMD_LINES, alp.getBinaryAOI())
        .seqControl(ALP_BITPLANE_LUT_MODE, ALP_BITPLANE_LUT_ROW)
        // .seqControl(ALP_BITPLANE_LUT_ENTRIES, not required)
        // LUT_ROW mode: the LUT size always matches the DMD height
        // (or AOI height, if enabled)
        .setBplut(bplutData.duplicate(alp.getBinaryAoiHeight()))
        // Duplicate: There must be one valid bitplane entry per active DMD row
        .setSeqScroll(1, 0, scrollToRow);
// ...
setTiming(alp, pictureTime, scrollToRow+1);       // calls AlpSeqTiming
```

Figure 22: Sequence setup for demoBinaryBplutRow()

5.5 Further consideration

5.5.1 Black fencing areas

The demonstrated scrolling over a long stripe of data shall display every image row on the exactly same number of DMD rows. If the program would just put the image beginning at the ALP sequence memory, then the upper rows would scroll out of the display after just a few frames.

Same applies to the lowest image rows, that scroll in for only one or a few frames.

The ALP BPLUT Sample introduces a concept of black fencing areas. These are two areas filled with dark pixels. They have a size according to the active DMD area (area of interest, ALP_SEQ_DMD_LINES).

With these fences, the scrolling sequence always starts and ends with a completely black frame and shows image data (“payload”) from the second to the last-but-one frame.

5.5.2 ALP_SEQ_CONFIG

Switching bit-planes for each DMD row is a challenging task for the ALP electronics. ALP sequences must be initialized in a special way for correct display and timing. The ALP API can be informed to do so using alp.devControl(ALP_SEQ_CONFIG, ALP_SEQ_CONFIG_ BITPLANE_LUT_ROW) before calling AlpSeqAlloc.

This can impose timing draw-backs in normal mode under some rare circumstances, so it is not the default ALP API setting. However, the ALP BPLUT Sample can use this setting generally.

5.5.3 Bitplane frequency

If required, the switching rate of bit-planes can safely be reduced in the bit-plane lookup table. The ALP BPLUT Sample supports this using the function CAlpBplut::distribute(). A source code change is required to do so (search for “distribute” and uncomment).

In demo case BPLUT_FRAME the resulting display has lower temporal bit-plane switching frequency. The effect on BPLUT_ROW is lower spatial switching. This could be required e.g., if larger scrolling steps are used (ALP_LINE_INC).

5.5.4 Active DMD size (Area of Interest)

The ALP frame rate can be increased by reducing the active DMD size. The ALP API supports this for binary display mode using ALP_SEQ_DMD_LINES.

The impact on scrolling-mode applications is a higher feed rate (substrate length per second), because feed rate is determined by frame rate * pixel size. Pixel size is usually fixed (depends on DMD native pixel size and the imaging optics magnification), but the frame rate can be tuned.

6 Conclusion

This application note theoretically shows how bit-plane lookup table modes can help to achieve high throughput in scrolling mode DMD applications. They still realize linear gray-scale doses, which renders them perfectly suitable for lithography applications, for example. ALP sequences can switch bit-planes either in high temporal or spatial frequency.

Now ViALUX has disclosed ALP API support for ALP_BITPLANE_LUT_MODE in all SuperSpeed V-Modules. The ALP BPLUT Sample application comprehensively shows all required device and sequence settings in source code.

Execution of this sample program demonstrates these modes on real-world V-Module hardware, providing a good impression about valid gray-scale dose generation in these use cases.

7 Table of Tables

8 Table of Figures