Application Note

Grabbing to a Host Buffer with Matrox Genesis-LC

June 12, 1998

Grabbing to a Host buffer

When grabbing to a Host buffer, Matrox Genesis-LC uses off-screen frame buffer memory (that is, frame buffer memory that is not configured for display purposes) as a FIFO, to buffer image data while the board waits for access to the PCI bus. Loss of image data could otherwise occur during long bus-access latencies, found in heavily loaded systems.

Matrox Genesis-LC uses hardware/software transfer control logic to double-buffer the grab between two temporary buffers set up in off-screen memory. This logic toggles the grab between one temporary buffer and the other, transferring grabbed data from the buffer in which data is not being grabbed. The double-buffering is done on a line-block basis rather than on a frame basis.

In general, the default size of the temporary buffers provides a good compromise between maintaining a small latency before the beginning of the transfer and minimizing the transfer's dependency on the PCI bus load. However, when the PCI bus is extremely loaded, the Matrox Genesis-LC board might not be able to maintain the grab and transfer sequence. If the request to transfer is not serviced in time, the grab might overwrite non-transferred data. This is known as a grab over-run.

For example, a typical load on the PCI bus is that of the Matrox Genesis-LC grabbing to a Host buffer while the Host is updating the Matrox Genesis-LC's display. When updating the GUI (in single-screen mode) and/or the display of grabbed and processed images, the display updates are frequent. This bi-directional traffic might increase the latency in servicing the request to write grabbed data to Host memory. At some point, the accumulated latencies might cause a grab over-run.

Optimizing the use of the frame buffers

If you experience grab over-runs, try the following solutions in the specified order:

1. Increase the default temporary buffer size. This should be enough to resolve your problems.
2. If possible, double-buffer the grab on a frame basis rather than a line-block basis. In this case, you have to manage the double-buffering of the grab yourself. This method involves one frame of latency before the beginning of the transfer.
3. Reduce the display resolution or, if possible, reduce the PCI bus load.

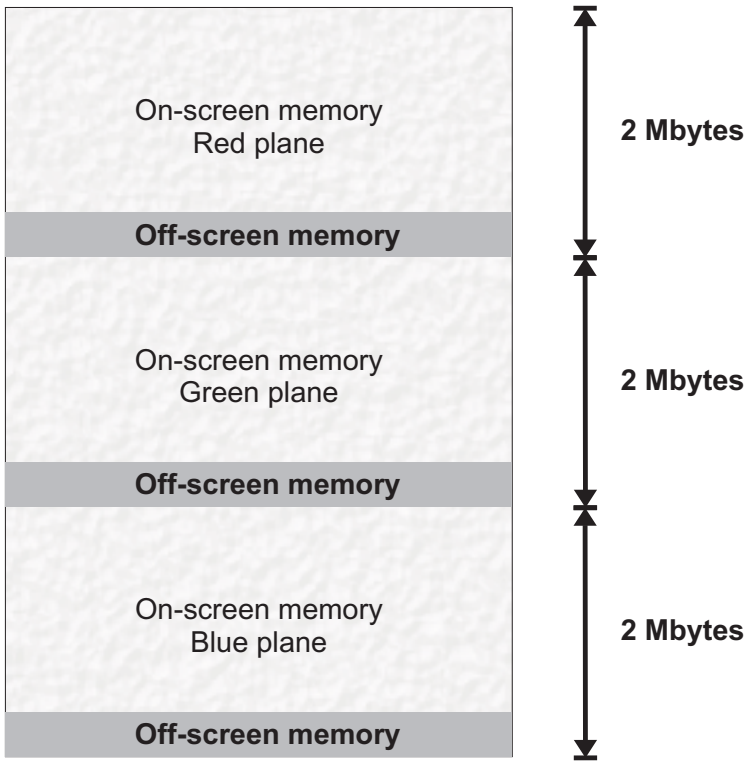
Increasing the default temporary buffer size

You can increase the default temporary buffer size. Larger temporary buffers make the grab's transfer-to-Host less dependent on the PCI bus load. That is, the grab is less affected by long bus-access latencies, found in heavily loaded systems, because more memory is available to buffer the data. Note, however, that larger temporary buffers also increase the latency before the beginning of the transfer.

The maximum size of each temporary buffer is limited by the amount of off-screen memory divided by three:

```
max temp buffer size = off-screen memory / 3
```

The amount of off-screen frame buffer memory is determined by the amount of frame buffer memory that is not configured for display purposes. Decreasing the display resolution increases the amount of off-screen memory.



The amount of off-screen memory per frame buffer plane is determined as follows:

off-screen memory/plane = 2 Mbytes - (w * h)

where *w* and *h* are the width and height of the selected display resolution.

The following table lists the amount of off-screen memory available for some typical display resolutions, as well as the maximum temporary buffer size for each resolution.

Display resolution	Bytes of off-screen memory/plane	Maximum temporary buffer size
640 x 480	1789952	596650
800 x 600	1617152	539050
1024 x 768	1310720	436906
1280 x 1024	786432	262144
1600 x 1200	177152	59050

You can increase the default temporary buffer size by adjusting the **SizeOfTmpBufferForHostGrab** field in the *genesis.ini* file.

Grabbing in on-board buffers

If increasing the size of the temporary buffers does not resolve your over-run problem, you can try double-buffering the grab on a frame basis rather than a line-block basis. The grab is then less dependent on the PCI bus load because the buffers don't have to be transferred as frequently and intermittent bus latencies are averaged over an entire frame. The downside of this process is that there is one frame of latency before the beginning of the transfer.

To implement this model, you have to manage the double-buffering. To do so, allocate two frame-sized buffers on-board and then grab a field/frame into one buffer while copying the other buffer to the Host. Since you control the synchronization between the grab and transfer, you can detect when something goes wrong.

❖ Note that, by default, the copy operation is driven by the Host CPU. To speed up the copy and reduce Host intervention, enable VIA-driven copies by setting the *MsysControl()* M_BUS_MASTER_COPY_TO_HOST control to M_ENABLE.

This method is only possible if sufficient on-board memory is available to allocate the two buffers.

Grabbing large frames of data

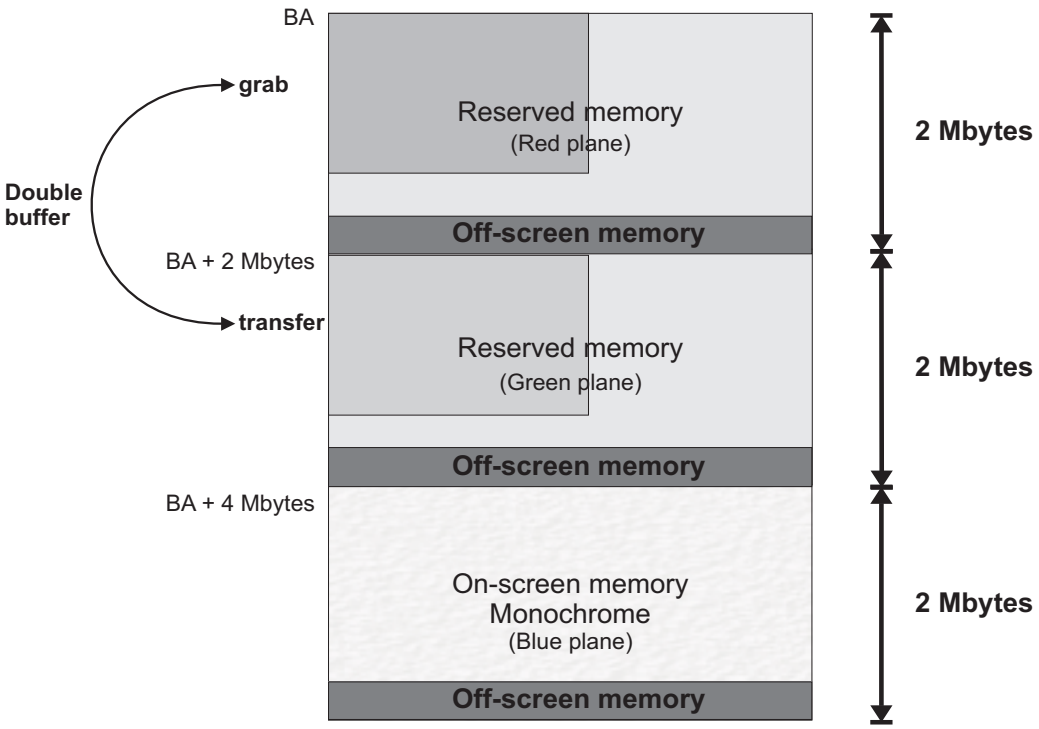
When grabbing large frames of data (for example, 2K X 2K and 4K X 4K), there is no way to store the whole frame in frame buffer memory. In this case, you can only grab to a Host buffer. However, if you are experiencing over-runs using this method, try the suggestions in the next subsection, *Other options*.

Monochrome display

When the frame buffers are in an 8-bit (monochrome) display resolution, an area of only one frame buffer plane is used to maintain the display. In this case, sufficient memory is available to keep two entire average-sized frames on-board.

Although an area of only one frame buffer plane is actually used to maintain the display, MIL reserves the corresponding area of the two other frame buffer planes so that you can switch between a monochome and color display without corrupting off-screen memory.

This means that if you try to allocate the buffers with an M_ON_BOARD attribute, there might not be sufficient true off-screen memory to allocate the entire buffer. Instead, you might need to allocate the grab buffers in the reserved frame buffer memory. To do so, inquire the address of the underlay frame buffer using *MsysInquire()*, and then, using *MbufCreate()*, create your two buffers at the appropriate offset (that is, 0, 2, or 4 Mbytes).



BA = Base address

If you allocate buffers in reserved display memory, you cannot switch to color mode or select a color buffer on the display; otherwise, the grab buffers will be corrupted.

An example

The following example shows how to allocate buffers in reserved display memory and then grab in these buffers.

```
/* File name: mgenlc.c
 * Synopsis:  This program allocates two grab buffers in the
 *            unused bands of the display and then grabs in them.
 *
 *            When the display is in monochrome mode, it
 *            uses only the blue plane and reserves the
 *            red and green planes. This example shows
 *            how to access this unused memory for grabbing
 *            purposes.
 *
 *           Note that the MsysControl() M_BUS_MASTER_COPY_TO_HOST
 *            control is only available in MIL 5.11 and later.
 */
/* Headers */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <mil.h>

/* Main function. */
void main(void)
{
  MIL_ID   MilApplication;
  MIL_ID   MilSystem     ;
  MIL_ID   MilDigitizer  ;
  MIL_ID   MilDisplay    ;
  MIL_ID   MilImageOnBoard[2];
  MIL_ID   MilImageDisp;
  void     *UnderlayPhysicalAddress = NULL;

 /* Allocations.*/
  MappAlloc(M_DEFAULT, &MilApplication);
  MsysAlloc(M_SYSTEM_GENESIS, M_DEF_SYSTEM_NUM, M_SETUP, &MilSystem);
  MdigAlloc(MilSystem, M_DEFAULT, M_DEF_DIGITIZER_FORMAT,
            M_DEFAULT, &MilDigitizer);
  MdispAlloc(MilSystem, M_DEFAULT, M_DEF_DISPLAY_FORMAT, M_DEFAULT,
             &MilDisplay);

 /* Force the display in monochrome mode on the blue plane. */
  MdispControl(MilDisplay, M_COLOR_MODE, M_BLUE);

 /* Allocate a display buffer. */
  MbufAlloc2d(MilSystem,
              (long)(MdigInquire(MilDigitizer, M_SIZE_X, M_NULL)),
              (long)(MdigInquire(MilDigitizer, M_SIZE_Y, M_NULL)),
              8L+M_UNSIGNED,
              M_IMAGE+M_PROC+M_DISP+M_NON_PAGED, &MilImageDisp);
  MbufClear(MilImageDisp, 0);

                                                                (cont...)
```

```
 /* Inquire the base address of the underlay plane. */
  MsysInquire(MilSystem, M_PHYSICAL_ADDRESS_UNDERLAY,
              &UnderlayPhysicalAddress);
 /* Create a buffer on the red plane (base address). */
  MbufCreateColor(MilSystem, 1,
                  (long)(MdigInquire(MilDigitizer, M_SIZE_X, M_NULL)),
                  (long)(MdigInquire(MilDigitizer, M_SIZE_Y, M_NULL)),
                  8L+M_UNSIGNED,
                  M_IMAGE+M_GRAB+M_PROC+M_ON_BOARD,
                  M_PHYSICAL_ADDRESS+M_PITCH_BYTE,
                  M_DEFAULT,
                  (void *)&(UnderlayPhysicalAddress),
                  &MilImageOnBoard[0]);

 /* Move to the green plane (base address+2MEG). */
  UnderlayPhysicalAddress = ((unsigned long) UnderlayPhysicalAddress) +
                            0x200000;
  MbufCreateColor(MilSystem, 1,
                  (long)(MdigInquire(MilDigitizer, M_SIZE_X, M_NULL)),
                  (long)(MdigInquire(MilDigitizer, M_SIZE_Y, M_NULL)),
                  8L+M_UNSIGNED,
                  M_IMAGE+M_GRAB+M_PROC+M_ON_BOARD,
                  M_PHYSICAL_ADDRESS+M_PITCH_BYTE,
                  M_DEFAULT,
                  (void *)&(UnderlayPhysicalAddress),
                  &MilImageOnBoard[1]);

 /* Enable bus master copies from the Genesis to the Host. */
  MsysControl(MilSystem, M_BUS_MASTER_COPY_TO_HOST, M_ENABLE);

/* Display the buffer. */
  MdispSelect(MilDisplay, MilImageDisp);
  printf("Press enter to grab into the first buffer\n");
  getchar();

 /* Grab and copy to the Host. */
  MdigGrab(MilDigitizer, MilImageOnBoard[0]);
  MbufCopy(MilImageOnBoard[0], MilImageDisp);
  printf("Press enter to grab into the second buffer\n");
  getchar();

 /* Grab and copy to the Host. */
  MdigGrab(MilDigitizer, MilImageOnBoard[1]);
  MbufCopy(MilImageOnBoard[1], MilImageDisp);
  getchar();

 /* Free allocations. */
  MbufFree(MilImageOnBoard[0]);
  MbufFree(MilImageOnBoard[1]);
  MbufFree(MilImageDisp);
  MdispFree(MilDisplay);
  MdigFree(MilDigitizer);
  MsysFree(MilSystem);
  MappFree(MilApplication);
}
```

Color display

When the frame buffers are in a 24-bit (color) display resolution, the three frame buffer planes are used to maintain the display. In this case, off-screen memory is the only memory available in which to perform double-buffering without affecting the display.

In a 1024 x 768 display resolution, there is 1310720 bytes of off-screen memory/plane available for a double buffering scheme. If, for example, your camera is a 512 x 480 RGB source, only 737280 (512 x 480 x 3) bytes are required to buffer one frame. So, you can allocate the buffers in the off-screen memory of two planes. Since you need to allocate the buffers in off-screen memory, simply allocate the buffers with *MbufAllocColor()* and use the M_ON_BOARD flag; the buffers will automatically be allocated in the appropriate off-screen memory.

When grabbing larger frames of data or when displaying in a higher resolution, the previous method might not be suitable. For example:

Display	Camera	Bytes missing in off-screen memory
1280 x 1024	1K x 1K mono source	262144
1600 x 1200	1K x 1K mono source	871424
1600 x 1200	512 x 480 mono source	68608

In these cases, the only way to achieve the grab is to use off-screen memory as FIFO and grab to a Host buffer. However, if you are experiencing over-runs using this method, try the suggestions in the following subsection.

Other options

In the unlikely event that the previous two suggestions do not resolve grab over-runs, you have the following options:

- You can try reducing the display resolution. This reduces the PCI bus load because there is less to update, and it increases the amount of off-screen memory so you can create larger temporary buffers.
- If your application permits, use other methods to reduce the PCI bus load. For example, reduce the number of display updates of the processed grabbed image; this, however, involves manual intervention.