AN11328

Implementing emWin's Multiple Buffering feature on the LPC1788 microcontroller

Rev. 1 — 11 February 2013 Application note

Document information

Info	Content
Keywords	emWin, graphics library, LPC1788, ARM Cortex-M3, multiple buffering, double buffering, triple buffering, LPCXpresso, Embedded Artists
Abstract	This application note describes the implementation of multiple buffering using Segger's emWin graphics library on the NXP LPC1788 Microcontroller.

Revision history

Rev	Date	Description
1	20130211	Initial version.

Contact information

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

This application note describes the multiple-buffering concept used in emWin. It also describes how to use the multiple-buffering feature on the NXP's LPC1788 ARM Cortex-M3 microcontroller. This application note can be used as an example to implement multiple buffering on other NXP microcontrollers containing a similar LCD controller.

1.1 Project overview

To display an image on the LCD screen, the content of the screen first needs to be drawn in the memory. To do this, we need a large contiguous piece of memory which can store the contents of the image pixel by pixel, called a framebuffer.

Multiple buffering is a method of using more than one buffer to eliminate screen artifacts when changing the content of the framebuffer. Because of the size of each buffer, these buffers usually reside in external memory. The LPC1788 microcontroller contains an external memory controller (EMC) as well as an LCD controller. The image is rendered to the external SDRAM and transferred to the TFT LCD using the EMC and the LCD controller.

1.2 System components and system setup

Fig 1 depicts the main system components required by this application. This application is developed and tested on Embedded Artists' LPC1788 Developer's Kit. The development kit contains Embedded Artists' LPC1788 OEM rev A board and rev PB1 of the OEM baseboard. The SDRAM used in this application contains multiple framebuffers. The LCD controller loads the frame from one of the buffers and sends it to a 7 inch TFT LCD.

The software included in this application note can be built with Keil µVision4 IDE, LPCXpresso5 IDE and IAR Embedded workbench IDE 4. For demonstration purposes, the application can also be simulated on a Windows PC by compiling it with Microsoft Visual C++ IDE, before building on another IDE and flashing it to the real target board. This project structure has been organized similar to the regular LPC1788 BSP released by NXP for emWin.

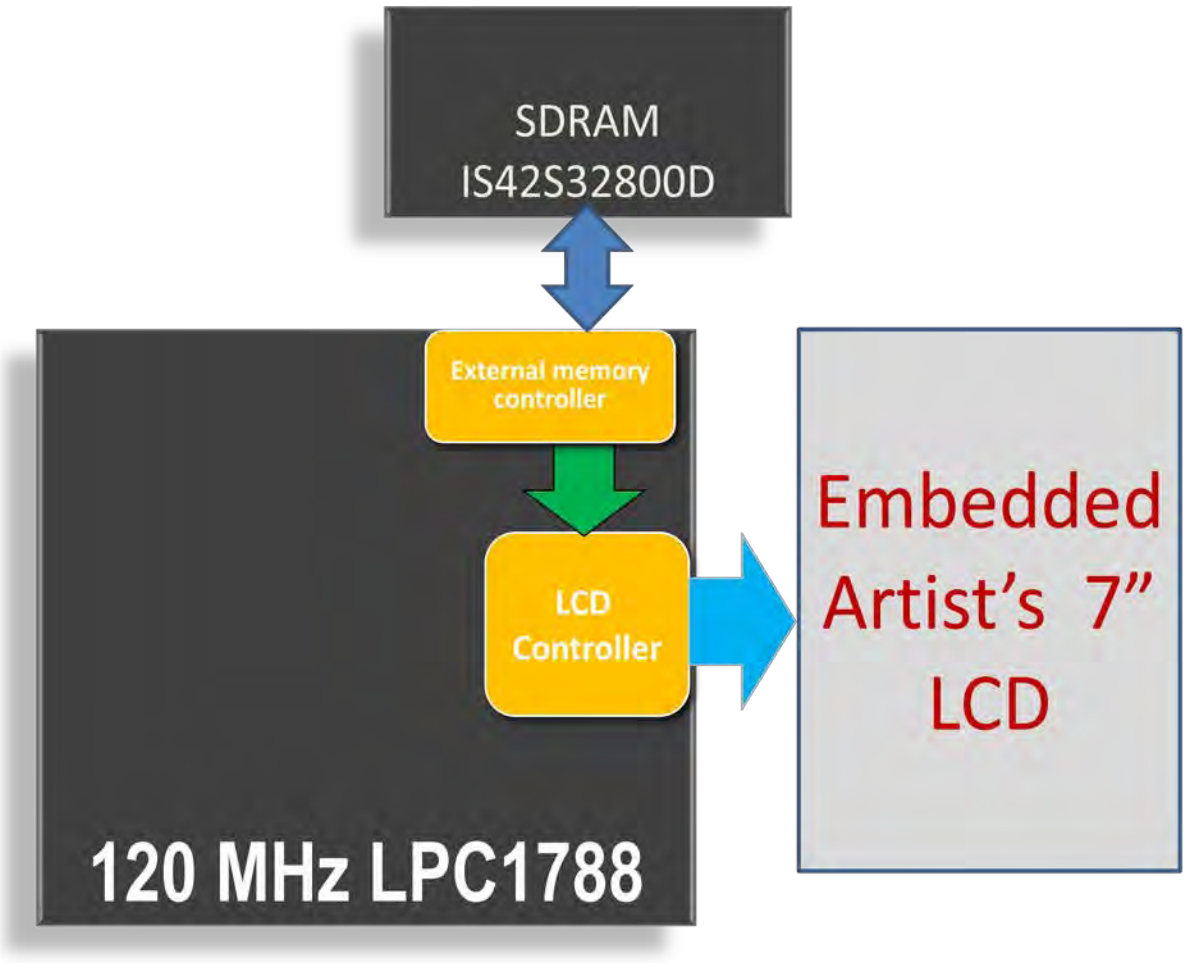

Fig 1. Main system components

2. Theory of operation

emWin renders a frame to a buffer which resides in the SDRAM, which is then sent to the LCD by the LCD controller. This buffer is called a framebuffer. The framebuffer is a contiguous block of memory which stores the contents of the LCD, pixel by pixel. In other words, this buffer contains a complete frame of data. During display of a frame, any rendering on this framebuffer can introduce screen artifacts which appear as flickering or tearing. Flickering artifacts occur when drawing and display is done from the same buffer at the same time. Tearing is the effect where display shows one part of the image from one frame and another part of the image from another frame.

To avoid these unwanted screen artifacts emWin can use more than one buffer. Multiple buffering is the use of more than one framebuffer, so that the display always shows a screen which is already completely rendered, even if a drawing operation is in progress. This way rendering and displaying can be done on two separate buffers. For emWin to use multiple buffering, this feature must be enabled.

Fig 2 shows single buffer implementation used by emWin with LPC1788 microcontroller.

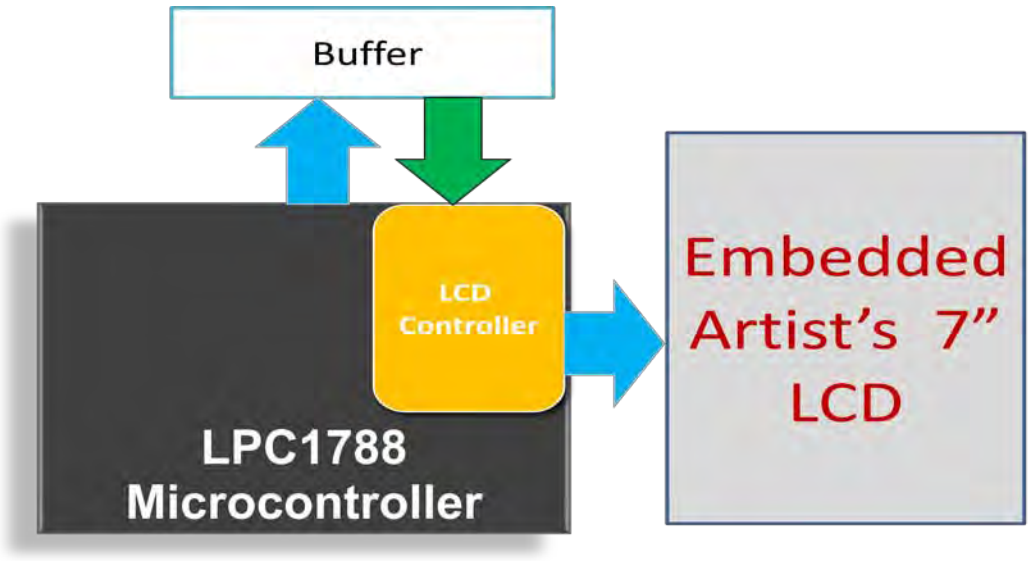

Fig 2. Single buffering: Same buffer used for drawing and display

With multiple buffers enabled, there is a front buffer which is used by the display controller to generate the picture on the screen, and one or more back buffers which are used for drawing operations. When starting the process of drawing, the current content of the front buffer is copied into one of the back buffers. After that, all drawing operations take place only on this back buffer.

After the drawing operation has been completed, the back buffer becomes the front buffer. Making the back buffer the visible front buffer normally only requires the modification of the framebuffer start address register of the display controller. Typically, the display is refreshed by the display controller about 60 times per second. This is called the refresh rate.

At the end of the current frame or at the beginning of an upcoming frame a synchronization signal is generated by the LCD controller to synchronize the refresh of the frames. This synchronization signal is called vertical synchronization signal (VSYNC), which was originally used in CRTs. The best moment to make the back buffer the new front buffer is this VSYNC signal. Tearing can occur if switching between buffers is performed during a normal display period, instead of a VSYNC period. Fig 3 shows the tearing screen artifact, which is introduced due to displaying one part of the image from one frame and the other part of the image from a different frame.

Fig 3. Tearing Effect

In general multiple buffering eliminates the following unwanted effects:

- The visible process of drawing a screen item by item
- Flickering effects caused by overlapping drawing operations
- Tearing effects caused by writing operations outside the vertical blanking period (only when swap of buffers is synchronized with the LCD's refresh)

The most popular buffering schemes used in emWin are double buffering and triple buffering. These schemes are explained below.

2.1 Double buffering

Double buffering uses two buffers, one front buffer and one back buffer. When a drawing operation starts, the current content of the front buffer is copied into the back buffer. The drawing operation is then executed upon the back buffer. After the drawing operation completes, the back buffer becomes the visible front buffer. For the best performance, the back buffer should become the front buffer as soon as a drawing operation completes, but it can introduce an image tearing artifact. As explained in the beginning of section 2, to prevent this artifact the best moment for making the back buffer the visible front buffer is reacting on the VSYNC signal of the display controller. Double buffering solves the problem of flickering but can show tearing if switching between buffers is not done during VSYNC period. Fig 4 shows double buffer implementation used by emWin with an LPC1788 microcontroller.

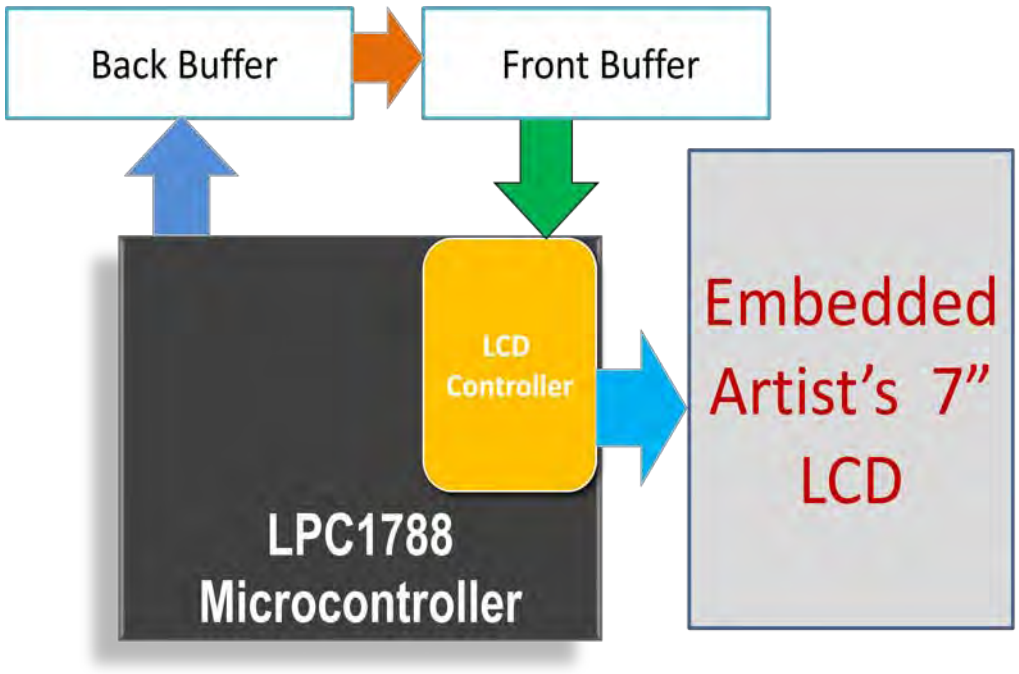

Fig 4. Double buffering: Separate buffers used for drawing and display

2.2 Triple buffering

Double buffering can resolve flickering effect but may not resolve tearing effect. Triple buffering is used to eliminate both the flickering and tearing effect.

In triple buffering three buffers are available: One front buffer and two back buffers. When a drawing operation starts, the current content of the front buffer is copied into the first back buffer. The drawing operation is then executed upon this back first buffer. After the drawing operation completes, the back buffer becomes the visible front buffer and the content of this buffer are copied to another buffer.

Contrary to the double buffer solution, it is not required to switch to the buffer immediately as there is one more buffer. Switching to the new front buffer could be done on the next VSYNC signal of the display controller which can be achieved by an interrupt service routine (ISR). Most of the display controllers which are able to deal with more than one framebuffer provide the VSYNC signal as an interrupt source. Within the ISR the pending front buffer should become visible. Until the pending front buffer becomes visible, it is not used for further drawing operations.

If a further drawing operation is initiated before the first back buffer has become visible, the second back buffer is used for the drawing operation. This way another drawing operation can be started without being stalled by waiting for the next VSYNC signal, thereby achieving a higher possible frame rate.

Fig 5 shows a triple buffer implementation used by emWin with LPC 1788 microcontroller.

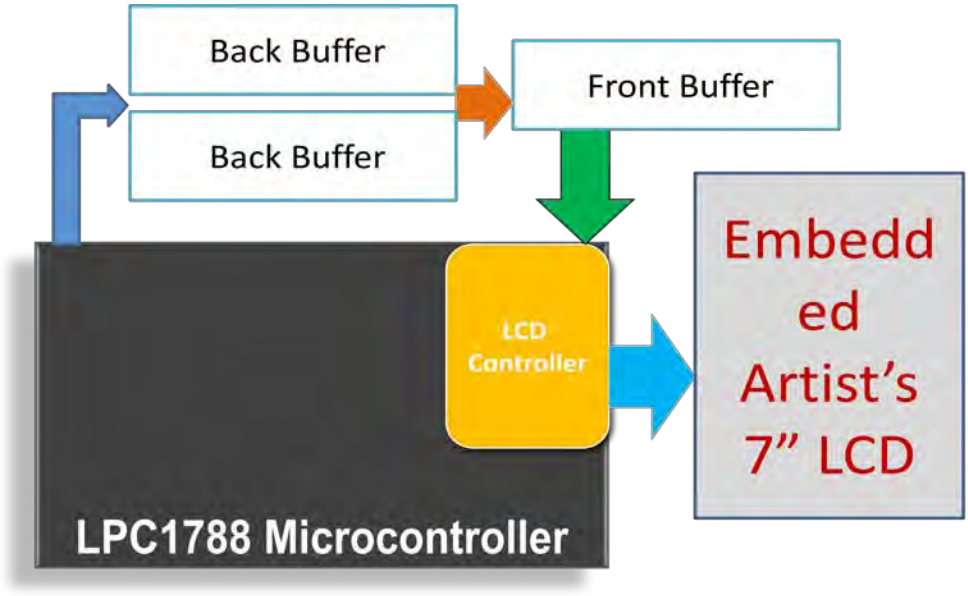

Fig 5. Triple buffering: Two separate buffer for drawing and one separate buffer for the display

2.3 Requirements

Following are the requirements to use multiple buffers:

- Enough video RAM for multiple framebuffers should be available.
- The display controller should support changing the framebuffer start address.
- Changing the display framebuffer address of the display controller needs to take place immediately, and it should be possible to do this on the VSYNC signal of the display controller.

Thankfully the LPC1788 microcontroller meets all three requirements in its hardware. On one hand, the LPC1788 microcontroller has an external memory controller which supports external SRAM as well as SDRAM; on the other hand it has an LCD controller with DMA capability. The LCD controller's upper panel DMA base address register must point to the correct framebuffer in the video RAM in order to eliminate screen artifacts.

In emWin, multiple buffering cannot be used with virtual screens.

3. Firmware overview

To implement multiple buffering in emWin, two functions in file LCDConf.c need to be modified:

- The display configuration routine **LCD_X_Config()**
- The driver callback function **LCD_X_DisplayDriver()**

The multiple-buffer interface must be configured before creating the display-driver device. This is normally done in LCD_X_Config(). The buffer numbers to be used are defined in file **multibuff.h,** and can be changed to a different value from the default value of 3.

As explained above, at the beginning of the drawing operation it is required to copy the content of the current front buffer to the back buffer. A simple memory copy can be used, or for better performance a DMA based routine. In this application a simple memory copy routine from emWin library is used. Fig 6 shows part of the code from the application.

```
/*********************************************************************
*
*       LCD_X_Config
*
* Purpose:
*   Called during the initialization process in order to set up the
*   display driver configuration.
*********************************************************************/
void LCD_X_Config(void) {
  const GUI_DEVICE_API * pDriver;
  GUI_MULTIBUF_Config(NUM_BUFFERS);

  #ifndef _WINDOWS
    _UpdateDisplayConfig();
  #endif
  //
  // Check framebuffer size
  //
  #ifndef _WINDOWS
    if ((FB_XSIZE * FB_YSIZE) < (VXSIZE_PHYS * VYSIZE_PHYS)) {
      while (1);  // Error, framebuffer too small
    }
  #endif
  //
  // Set display driver and color conversion for 1st layer
  //
  #ifndef _WINDOWS
  if (_Display == DISPLAY_TRULY_240_320) {
    pDriver = DISPLAY_DRIVER_TRULY;
  } else {
    pDriver = DISPLAY_DRIVER_OTHER;
  }
  #else
  pDriver = GUIDRV_WIN32;
  #endif
  GUI_DEVICE_CreateAndLink(pDriver, COLOR_CONVERSION, 0, 0);
  #ifndef _WINDOWS
  LCD_SetDevFunc(0, LCD_DEVFUNC_COPYBUFFER, (void (*)())_LCD_CopyBuffer);
  #endif
  //
  // Display driver configuration, required for Lin-driver
  //
```

Fig 6. LCD_X_Config code

The copy buffer function is shown in Fig 7.

```
/*********************************************************************
*
*                          _LCD_CopyBuffer
*
Purpose: It copy current front buffer to back buffer
**********************************************************************/

static void _LCD_CopyBuffer(int LayerIndex, int IndexSrc, int IndexDst) {
  unsigned long BufferSize, AddrSrc, AddrDst;
  //
  // Calculate the size of one frame buffer
  //
  BufferSize = XSIZE_PHYS * YSIZE_PHYS * PIXEL_WIDTH;
  //
  // Calculate source- and destination address
  //
  AddrSrc = VRAM_ADDR_PHYS  + BufferSize * IndexSrc;
  AddrDst = VRAM_ADDR_PHYS  + BufferSize * IndexDst;
  GUI_MEMCPY((void *)AddrDst, (void *)AddrSrc, BufferSize);
}
```

Fig 7. Copy buffer function

After the drawing process has been completed, the back buffer needs to become visible. The display driver sends a LCD_X_SHOWBUFFER command to the display driver callback function. The callback function then has to react on the command and should make sure that the buffer becomes visible. This can be done either by an ISR or by directly writing the right address into the framebuffer start address of the display controller. In the LPC1788 LCD controller's, LCD base update interrupt can be used for this purpose. As current base update registers in LPC1788s LCD controller are updated at vertical synchronization, LCD base registers can be updated directly with framebuffer without using ISR. This application is made to be used with ISR and without ISR. ISR can be used by defining MULTIBUFF_USE_ISR 1 in multibuff.h file as shown below.

```
/* When 1 use ISR otherwise use direct update of base update register*/
#define MULTIBUFF_USE_ISR   1
```

Fig 8. MULTIBUFF_USE_ISR

Following routine shows the initialization of the base-update interrupt. LPC1788 can drive STN dual panel as well as TFT. To support Dual panel, it has two base update registers: LCDUPBASE and LCDLPBASE register. The LCD next base address update interrupt asserts when either the LCDUPBASE or LCDLPBASE values have been transferred to the LCDUPCURR or LCDLPCURR registers respectively. In case of Embedded Artists' 7" TFT, only LCDUPBASE and LCDUPCURR are used. In this application example, the LCD base address update interrupt asserts when LCD base register (LCDUPBASE) value has been transferred to current base register (LCDUPCURR). This signals that it is safe to update the LCD base address register with new frame base address which may be required in some of the screen animation. For more detail about NXP's LCD Controller, please have a look at the LPC178x/7x user manual.

As mentioned in case of TFT, only LCDUPBASE register has significance. The base update interrupt is implemented as shown in below code.

```
#if MULTIBUFF_USE_ISR
/*********************************************************
* LCD IRQ Handler
* This is base register update interrupt
*********************************************************/

static int32_t  _PendingBuffer;

void LCD_IRQHandler(void) {
  unsigned long Addr, BufferSize;
  //Check of base update interrupt
  if((LPC_LCD->INTSTAT)&(1<<2)){
    if (_PendingBuffer >= 0) {
    //
    // Calculate address of the given buffer
    //
      BufferSize = XSIZE_PHYS * YSIZE_PHYS * PIXEL_WIDTH;
      Addr = VRAM_ADDR_PHYS + BufferSize *_PendingBuffer;
      //
      // Make the given buffer visible
      //
      LPC_LCD->UPBASE = Addr;
      //
      // Send a confirmation that the buffer is visible now
      //
      GUI_MULTIBUF_Confirm(_PendingBuffer);
      _PendingBuffer = -1;
    }
  LPC_LCD->INTCLR |= 1<<2;
  }
}
#endif
```

Fig 9. Base update interrupt

When used with ISR, function LCD_X_DisplayDriver shows the pending buffer. This pending buffer can be switched in the ISR as shown in Fig 9.

When the ISR is not used, function LCD_X_DisplayDriver is responsible to switch the buffer.

The implementation of this function in both cases, with ISR and without ISR, is shown in Fig 10 and Fig 11.

```
int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd, void * p) {
  int r;
#ifndef _WINDOWS
  #if (!MULTIBUFF_USE_ISR)
  unsigned long Addr, BufferSize;
  #endif
#endif
  (void) LayerIndex;
  GUI_USE_PARA(LayerIndex);
  switch (Cmd) {
  case LCD_X_INITCONTROLLER:
    //
    // Called during the initialization process in order to set up the
    // display controller and put it into operation. If the display
    // controller is not initialized by any external routine this needs
    // to be adapted by the customer...
    #ifndef _WINDOWS
      _InitController(LayerIndex);
    #endif
    return 0;
 case LCD_X_SETVRAMADDR: {
    LCD_X_SETVRAMADDR_INFO * pData;
    pData = (LCD_X_SETVRAMADDR_INFO *)p;
   #ifndef _WINDOWS
    LPC_LCD->UPBASE = (U32)pData->pVRAM;
   #endif
   }
 break;
 case LCD_X_SETORG:{
   //
   // Required for setting the display origin which is passed in the 'xPos' and 'yPos' element of p
   //
   #ifndef _WINDOWS
    LCD_X_SETORG_INFO * pData;
    LCD_X_SETVRAMADDR_INFO Info;

    pData = (LCD_X_SETORG_INFO *)p;
    Info.pVRAM = (void *)(VRAM_ADDR_PHYS + (pData->yPos * XSIZE_PHYS * PIXEL_WIDTH));
    LCD_X_DisplayDriver(LayerIndex, LCD_X_SETVRAMADDR, &Info);

   #endif
   return 0;
 }
  case LCD_X_SHOWBUFFER: {
   LCD_X_SHOWBUFFER_INFO * pData;
   pData = (LCD_X_SHOWBUFFER_INFO *)p;

   //
   // Remember buffer index to be used by ISR
   //
```

Fig 10. LCD_X_DisplayDriver (1 of 2)

```
    #ifndef _WINDOWS
      #if MULTIBUFF_USE_ISR
      _PendingBuffer = pData->Index;
      #else
      //
      // Calculate address of the given buffer
      //
      BufferSize = BufferSize = XSIZE_PHYS * YSIZE_PHYS * PIXEL_WIDTH;
      Addr = VRAM_ADDR_PHYS + BufferSize * pData->Index;
      //
      // Make the given buffer visible
      //
      LPC_LCD->UPBASE = Addr;
      //
      // Send a confirmation that the buffer is visible now
      //
      GUI_MULTIBUF_Confirm(pData->Index);
      #endif
    #endif
    }
  return 0;
  default:
    r = -1;
  }
  return r;
```

Fig 11. LCD_X_DisplayDriver (2 of 2)

All above code is part of the driver and initialization, whereas below functions need to be called in the user's application. Function **GUI_MULTIBUF_Begin()** needs to be called before the drawing operation and **GUI_MULTIBUF_End()** needs to be called after completion of the drawing operation.

```
    /* call this function before drawing */
    GUI_MULTIBUF_Begin();
    /*Draw image */
    GUI_DrawGradientH(0, 0, LCD_X_SIZE, LCD_Y_SIZE, GUI_RED, GUI_BLUE);
    GUI_DrawBitmapMag(&bmLPCWarewebbanner160x90,0,0, magx, magy);
    /* call this function after drawing */
    GUI_MULTIBUF_End();
```

Fig 12. MultiBufferingTest.c

Fig 12 shows a code snippet in file MultiBufferingTest.c which magnifies and displays an image (array bmLPCWarewebbanner160x90[], inside LPCWare-web-banner-160x90.c file) using multiple buffers. Bitmap converter utility from emWin is used to convert the image from ..\Start\image\LPCWare-web-banner-160x90.bmp to c file. The bitmap converter utility can be found in the emWin BSP inside folder of ..\Start\Tools\.

The Window Manager (WM) is able to use the multiple-buffer feature automatically. The function **WM_MULTIBUF_Enable()** can be used to enable this functionality. If enabled the WM first switches to the back buffer before redrawing the invalid windows. After drawing all invalid windows, the new screen becomes visible. This hides the process of drawing a screen window by window. To enable multiple buffers just pass parameter 1 to above function.

Fig 13 shows the system setup. You can power the LPC1788 board by connecting the mini USB connector J25 to a PC or USB hub. Alternatively, a 5 V DC adapter can be used to power the Embedded Artists' board and the LCD after plugging it into connector J24. A Keil uLink-2, uLink-Pro, uLink ME or other supported JTAG debuggers will be needed with Keil µVision4 IDE to flash and debug the software. Jlink is used to debug with IAR Embedded Workbench. LPCLink is used to debug with LPCXpresso. The Embedded Artists' 7 inch LCD is connected to connector J26 of the LPC1788 base board. For details on the connection please check the Embedded Artists' LPC1788 Developers Kit User Guide and the LCD Board User Guide.

Fig 13. Demo setup

4. Expected output

The project can be built by using any of the following IDEs:

- Keil uVision 4
- IAR Embedded Workbench 6
- LPCXpresso 5

For the exact version of the toolchain please go through the project readme file. After flashing and running this application example, an LPCware image as shown in Fig 13 will appear. Fig 13 shows one of the magnified images. The animation will continuously run with different magnification ratios. After the entire magnification sequence has been displayed, it will come back to its original size. When you change NUM_BUFFERS to 1 you can see flickering during magnification. In this case emWin is using only one buffer.

This demo can also be simulated using the Microsoft Visual C++ IDE which is useful when final target is not ready.

5. Conclusion

emWin's multiple-buffering feature can be used to eliminate the effects of flickering and tearing. With NXP's LPC1788 microcontroller, the multiple-buffering feature can be used with or without ISR. NXP's LPC1788 and many other LPC microcontrollers can be connected to external SRAM or SDRAM to fulfill the memory requirement of the multiple buffering feature. The Base Update interrupt in the NXP LPC1788 microcontroller ensures that the Base Address Registers are updated only when values from these registers have been transferred from the Current Base Address register.

6. Legal information

6.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

6.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

6.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

7. List of figures

8. Contents

For more information, visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 11 February 2013
Document identifier: AN11328