# AN11218

## emWin Porting guide: EA LPC1788 BSP to Keil MCB1700

Rev. 1 — 21 June 2012 Application note

**Document information**

| Info | Content |
|---|---|
| **Keywords** | emWin, Graphical, LCD, MCB1700, LPC1768, Porting |
| **Abstract** | This application note illustrates how to port an existing emWin BSP to another board, assembled with another type of MCU and another type of LCD. This guide shows step-by-step how to do this by porting the Embedded Artists' EA1788 (with on-chip LCD controller) BSP to Keil's MCB1700 (without on-chip LCD controller) board. |

## Revision history

| Rev | Date | Description |
|---|---|---|
| 1 | 20120621 | 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

NXP offers emWin Board Support Packages (BSPs) for two boards: Embedded Artists' EA1788 and Keil's MCB1700. Although more boards will be supported in the future, many times it is required to port emWin to another target, e.g., custom hardware. EmWin is designed to be easily portable and this guide shows the required steps to do this.

This guide shows step-by-step how the EA1788 BSP was ported to the MCB1700 board. Table 1 shows the most important differences between the two boards.

**Table 1. Comparison between the EA1788 and the MCB1700 board**

| | **EA1788** | **MCB1700** |
|---|---|---|
| **MCU** | LPC1788 | LPC1768 |
| **On-chip LCD controller** | Yes | No |
| **On-board ROM** | 128 MB + 512 kB | 512 kB |
| **On-board RAM** | 32 MB + 96 kB | 64 kB |
| **LCD** | 3.2", 240 x 320 pixels/ 4.3", 480 x 272 pixels/ 7", 800 x 480 pixels | 2.4", 240 x 320 pixels |
| **Interface to LCD** | 16-bit parallel | SPI |

This step-by-step guide assumes the use of the LPCXpresso IDE, though the steps are identical if using Keil or IAR. This chapter continues by explaining the steps involved in porting emWin and explains the software organization of the EA1788 BSP. The following chapters show the steps taken to successfully port the EA1788 project to the MCB1700 project.

## 1.1 Steps involved in porting emWin

Four major steps can be distinguished when porting emWin:

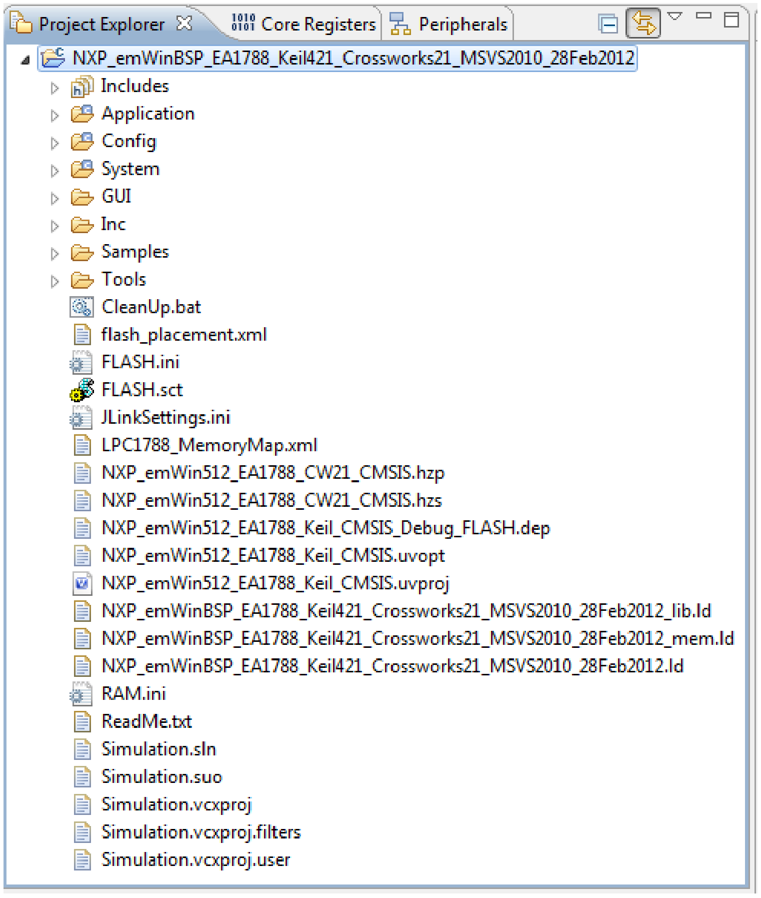
1. Change MCU-specific settings / source code.
2. Change board-specific settings / source code.
3. Change LCD related settings /source code.
4. Change emWin related settings / source code.

## 1.2 Software organization of the EA1788 BSP

The BSP used in this guide as a base for porting is available for download as a zip file at http://www.lpcware.com/content/project/emwin-graphics-library, under the following name: "emWin 5.14 BSP for Embedded Artist's EA1788 board and the 3.2", 4.3"and 7" LCDs - LPCXpresso 4, Keil uVision 4, IAR Workbench 6.22, Visual Studio 2010".

The zip file contains a single executable file, called "NXP_emWin514_BSP.exe". When this executable is run it will extract another zip file, called "NXP_emWin514_BSP.zip". After extracting this zip file, two folders are added to the file-system: "Start" & "Doc".

The "Start" folder can be imported into LPCXpresso. After LPCXpresso is done importing, the files displayed in Fig 1 are visible in the Project Explorer tab.



```
NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012
  Includes
  Application
  Config
  System
  GUI
  Inc
  Samples
  Tools
  CleanUp.bat
  flash_placement.xml
  FLASH.ini
  FLASH.sct
  JLinkSettings.ini
  LPC1788_MemoryMap.xml
  NXP_emWin512_EA1788_CW21_CMSIS.hzp
  NXP_emWin512_EA1788_CW21_CMSIS.hzs
  NXP_emWin512_EA1788_Keil_CMSIS_Debug_FLASH.dep
  NXP_emWin512_EA1788_Keil_CMSIS.uvopt
  NXP_emWin512_EA1788_Keil_CMSIS.uvproj
  NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012_lib.ld
  NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012_mem.ld
  NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012.ld
  RAM.ini
  ReadMe.txt
  Simulation.sln
  Simulation.suo
  Simulation.vcxproj
  Simulation.vcxproj.filters
  Simulation.vcxproj.user
```

**Fig 1. File and directory structure of the EA1788 BSP**

The following is a description of the folders in the Project Explorer tab:

- **Application**. This folder contains a number of sample applications. The fairly simple “GraphXYDemo” is set as active demo.
- **Config**. This folder holds several emWin configuration files to configure the emWin GUI, how emWin is supposed to control the LCD, and how it can control the cursor (e.g. by touch screen, joystick, mouse, if any are present).
- **System**. All hardware specific functions and configurations are present in this folder. It contains the CMSIS files, start-up files and driver software required by the board.
- **GUI**. The GUI folder contains all emWin related files. These are the pre-compiled libraries and the emWin header files.
- **Samples**. This folder contains a large number of sample files. It contains sample applications, sample configurations and sample drivers.
- **Tools**. Contains a number of windows executable files which can come in handy when developing with emWin, e.g. several conversion programs, a GUI builder and a VNC viewer.

Only the “Application”, “Config” and “System” folders are needed when porting emWin.

## 1.3 Getting started

Before changing any of the files, it is recommended to make a full copy of the original EA1788 project within the LPCXpresso workspace. Having the original BSP in the workspace can come in handy when debugging the new BSP by comparing the two BSP's to each other.

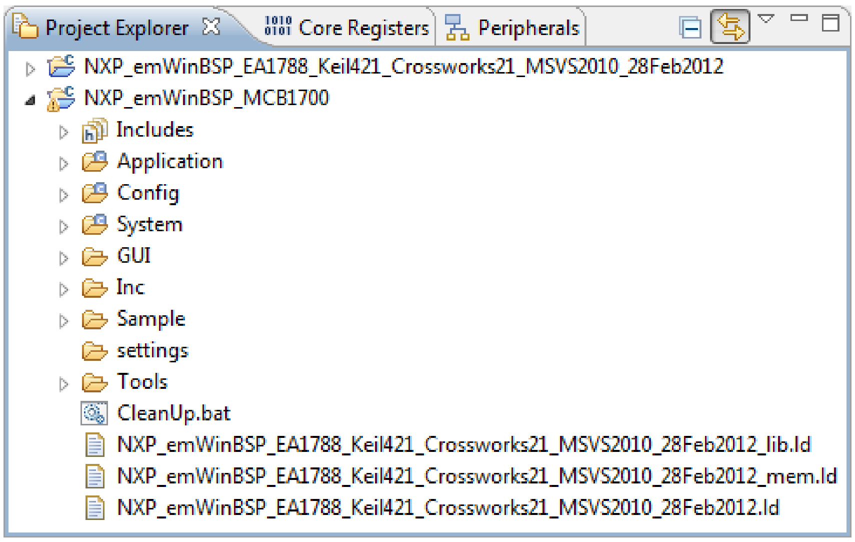
The resulting copy is renamed to "NXP_emWinBSP_MCB1700".

Depending on which compiler has to be supported, files may be deleted in order to keep a tidy workspace. If a particular compiler does not need to be supported, the files shown in the below list may be deleted.

Table 2. List of files

| File name | Development platform |
|---|---|
| flash_placement.xml | [ CrossWorks ] |
| LPC1788_MemoryMap.xml | [ CrossWorks ] |
| NXP_emWin512_EA1788_CW21_CMSIS.hzp | [ CrossWorks ] |
| NXP_emWin512_EA1788_CW21_CMSIS.hzs | [ CrossWorks ] |
| FLASH.ini | [ Keil ] |
| FLASH.sct | [ Keil ] |
| JLinkSettings.ini | [ Keil ] |
| NXP_emWin512_EA1788_Keil_CMSIS_Debug_FLASH.dep | [ Keil ] |
| NXP_emWin512_EA1788_Keil_CMSIS.uvopt | [ Keil ] |
| NXP_emWin512_EA1788_Keil_CMSIS.uvproj | [ Keil ] |
| RAM.ini | [ Keil ] |
| System/HW/Flash.icf | [ Keil ] |
| Application/cr_startup_lpc178x.c | [ LPCXpresso ] |
| NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012_lib.ld | [ LPCXpresso ] |
| NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012_mem.ld | [ LPCXpresso ] |
| NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012.ld | [ LPCXpresso ] |
| Simulation.sln | [ Visual Studio ] |
| Simulation.suo | [ Visual Studio ] |
| Simulation.vcxproj | [ Visual Studio ] |
| Simulation.vcxproj.filters | [ Visual Studio ] |
| Simulation.vcxproj.user | [ Visual Studio ] |

For this porting example only LPCXpresso is supported. All files related to the other compilers are removed from the project. After deleting those files, the workspace should look similar as in Fig 2.



Project Explorer | Core Registers | Peripherals

```
▷ NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012
◢ NXP_emWinBSP_MCB1700
    ▷ Includes
    ▷ Application
    ▷ Config
    ▷ System
    ▷ GUI
    ▷ Inc
    ▷ Sample
      settings
    ▷ Tools
      CleanUp.bat
      NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012_lib.ld
      NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012_mem.ld
      NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012.ld
```

**Fig 2. Result after copying, renaming and removing unnecessary files**

## 2. Step 1: Change MCU-specific settings / source code

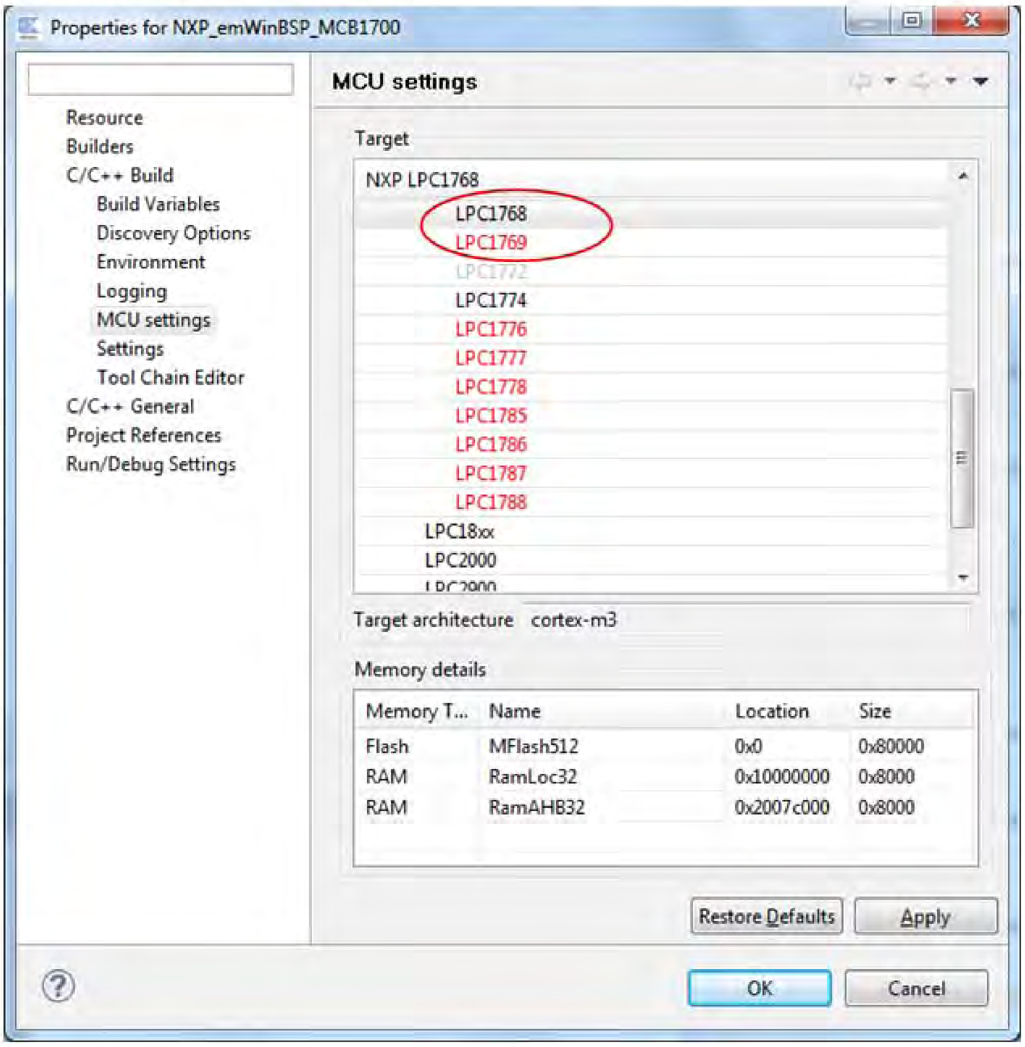
All functions and settings related to the controller should be changed to support the LPC1768 instead of the LPC1788.

### 2.1 Change project settings

Though this step depends on the IDE used, the general idea behind this step remains the same. Each compiler/IDE has certain project settings defining which target the software should run. For porting from the EA1788 to the MCB1700, these settings need to be changed to support the LPC1768 controller.

#### 2.1.1 Changing the target

In LPCXpresso these settings can be changed by opening the project properties window (Project Explorer – Right-click on the MCB1700 project – Choose “properties”). Currently the targeted MCU is the LPC1788; this should be changed into the LPC1768. Fig 3 shows the setting that has to be changed.



MCU settings

Target

NXP LPC1768
LPC1768
LPC1769
LPC1772
LPC1774
LPC1776
LPC1777
LPC1778
LPC1785
LPC1786
LPC1787
LPC1788
LPC18xx
LPC2000
LPC2900

Target architecture cortex-m3

Memory details

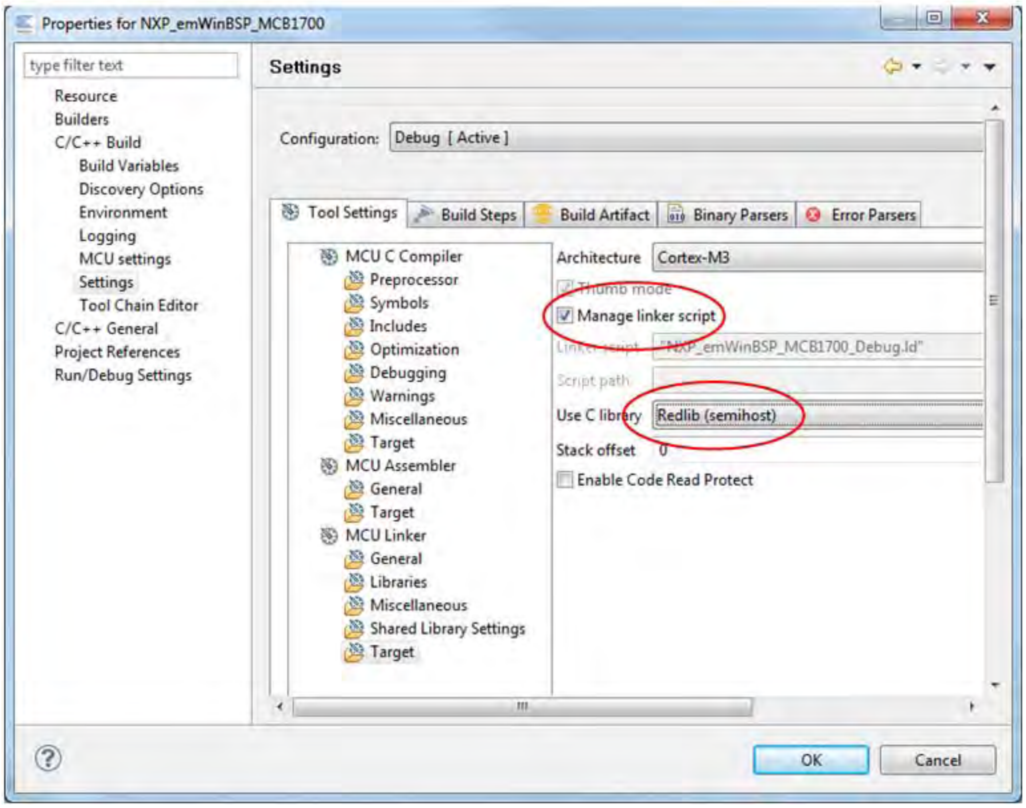
| Memory T... | Name | Location | Size |
|---|---|---|---|
| Flash | MFlash512 | 0x0 | 0x80000 |
| RAM | RamLoc32 | 0x10000000 | 0x8000 |
| RAM | RamAHB32 | 0x2007c000 | 0x8000 |

**Fig 3. MCU settings tab in properties window changed to the LPC1768**

## 2.1.2 Changing the linker settings

As shown in Table 1, the EA1788 board has external RAM and external flash memory. To support these external memories, the standard linker script is changed to support external memory for the EA1788 BSP. Because the MCB1700 board does not have these external memories, the linker script must be changed accordingly.

This can be done by opening the linker settings. These settings can be found in the project properties at “Settings” – “Tool Settings” – “MCU Linker” – “Target” (Fig 4). As the standard LPC1768 linker script is sufficient for the MCB1700, the checkbox “Manage linker script” can be checked, thereby restoring the default settings. Checking this box also defaults the “Use C library” settings to “Redlib (none)”. However, emWin needs to dynamically allocate memory, thus the “Use C Library” setting should be changed to “Redlib (semihost)” (for using semihosting) or to “Redlib (nohost)” (for no semihosting).



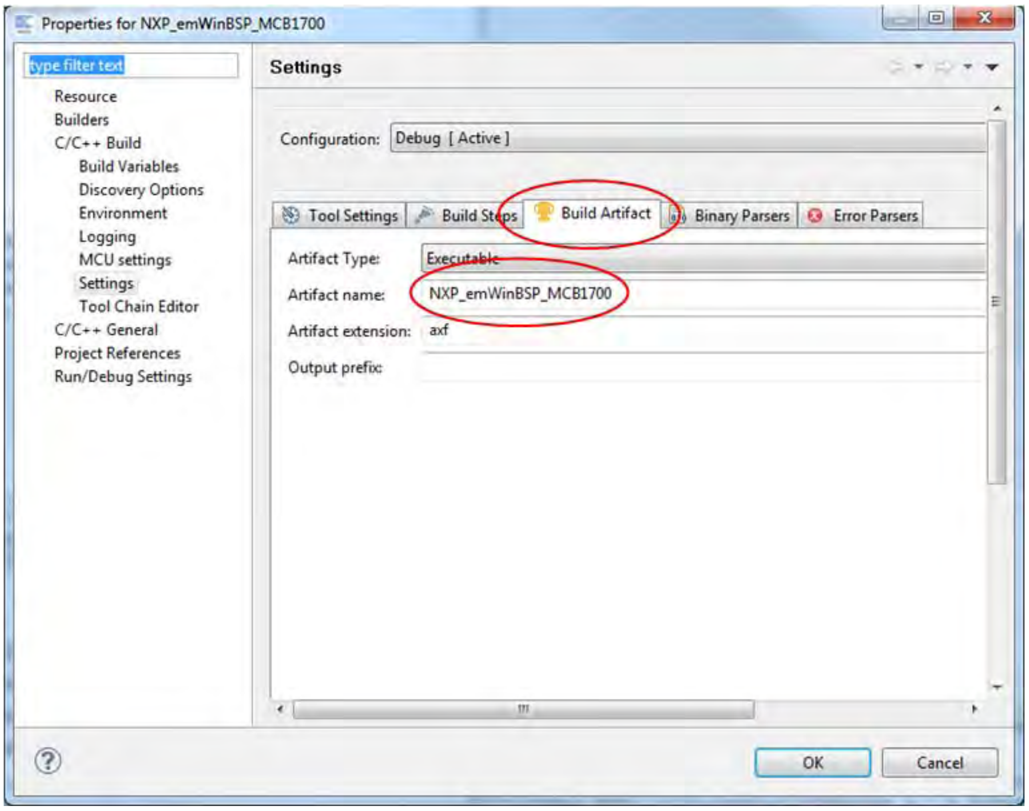
Configuration: Debug [ Active ]

Architecture: Cortex-M3
Thumb mode
[✓] Manage linker script
Linker script: "NXP_emWinBSP_MCB1700_Debug.ld"
Script path:
Use C library: Redlib (semihost)
Stack offset: 0
[ ] Enable Code Read Protect

**Fig 4. Linker settings changed to be managed by IDE, C library is set to Redlib (semihost)**

## 2.1.3 Changing the name of the Build Artifact

The last remaining thing to change is the name of the Build Artifact (= generated executable). The name in the EA1788 BSP is set to “NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012”. To change the name to “NXP_emWinBSP_MCB1700”, the “Artifact name” needs to be changed. This property can be found in the project properties at “Settings” – “Build Artifact” tab (Fig 5).


Properties for NXP_emWinBSP_MCB1700

Settings

Configuration: Debug [ Active ]

Build Artifact

Artifact Type: Executable

Artifact name: NXP_emWinBSP_MCB1700

Artifact extension: axf

Output prefix:

Fig 5. Build Artifact name set to NXP_emWinBSP_MCB1700

Three additional files (linker scripts for the EA1788) may now be removed from the project by removing them in the “Project Explorer” window:

NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012_lib.ld

NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012_mem.ld

NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012.ld

## 2.2 Change controller related source code and header files.

The next step is to change the source code and the header files to support the LPC1768 and remove all references to the LPC1788.

### 2.2.1 Change the CMSIS device header file

Remove the “LPC177x_8x.h” (in folder: “System/HW/DeviceSupport/”) from the project explorer and replace it with the correct header file (“LPC17xx.h”). These header files can be found at http://www.lpcware.com/.

### 2.2.2 Change the system_LPC file

Remove the “system_LPC177x_8x.c” and “system_LPC177x_8x.h” files (in folder: “System” – “HW” – “DeviceSupport”) from the project explorer and replace it with the “system_LPC17xx.c” and “system_LPC17xx.h” files.

### 2.2.3 Change the startup file

Remove the code red startup file “cr_startup_lpc178x.c” file (in folder: “Application”) and replace it with the “cr_startup_lpc176x.c” file.

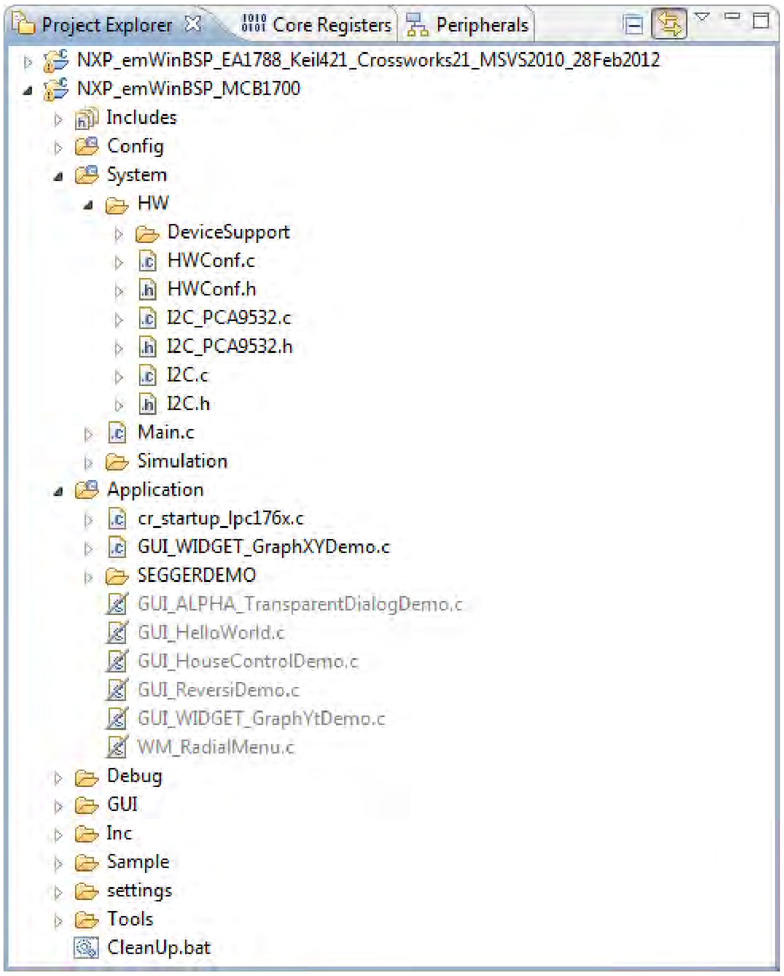
If IAR and Keil support are not necessary, the files “startup_LPC177x_8x_IAR.s” and “startup_LPC177x_8x_Keil.s” (in folder: “System” – “HW” – “DeviceSupport”) may be removed. If IAR and Keil support is required, these two files should be replaced with the LPC1768 IAR and Keil startup files.

### 2.2.4 Change CMSIS files

When porting from the EA1788 to the MCB1700 it is not necessary to change the CMSIS files (“core_cm3.c”, “core_cm3.h”, “core_cmFunc.h”, “core_cmInstr.h”) as both the LPC1788 and the LPC1768 have a Cortex M3 core. When porting to another Cortex M-architecture, it is necessary to update these CMSIS files.

**Note:** When porting to another architecture, it is necessary to use other emWin libraries matching the targeted architecture. The libraries supplied with the EA1788 and the MCB1700 BSPs are compiled for the Cortex M3 architecture. The zip file of Pre-compiled libraries (which can be found on http://www.lpcware.com/content/project/emwin-graphics-library) contains libraries for ARM7, ARM9, M0, M3, and M4. There are two sets of these five binaries: one built with LPCXpresso and the other built with IAR’s EWARM. The latter can be used with the Keil compiler by simply changing the extension of the files from .a to .lib. Binaries built with Rowley’s Crossworks are currently not available.

When all steps are completed, the files in the project explorer should look similar as in Fig 6.

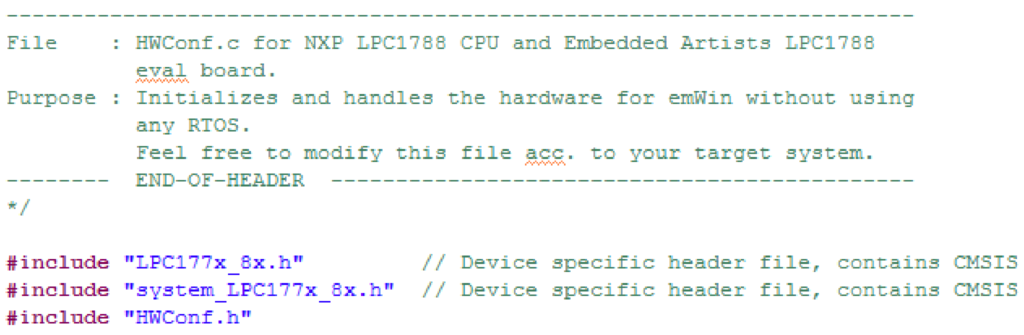


```
▷ NXP_emWinBSP_EA1788_Keil421_Crossworks21_MSVS2010_28Feb2012
◢ NXP_emWinBSP_MCB1700
    ▷ Includes
    ▷ Config
    ◢ System
        ◢ HW
            ▷ DeviceSupport
            ▷ HWConf.c
            ▷ HWConf.h
            ▷ I2C_PCA9532.c
            ▷ I2C_PCA9532.h
            ▷ I2C.c
            ▷ I2C.h
        ▷ Main.c
        ▷ Simulation
    ◢ Application
        ▷ cr_startup_lpc176x.c
        ▷ GUI_WIDGET_GraphXYDemo.c
        ▷ SEGGERDEMO
          GUI_ALPHA_TransparentDialogDemo.c
          GUI_HelloWorld.c
          GUI_HouseControlDemo.c
          GUI_ReversiDemo.c
          GUI_WIDGET_GraphYtDemo.c
          WM_RadialMenu.c
    ▷ Debug
    ▷ GUI
    ▷ Inc
    ▷ Sample
    ▷ settings
    ▷ Tools
      CleanUp.bat
```

Fig 6. Project explorer after changing the controller related files

### 2.2.5 Changing in-source references to the LPC1788

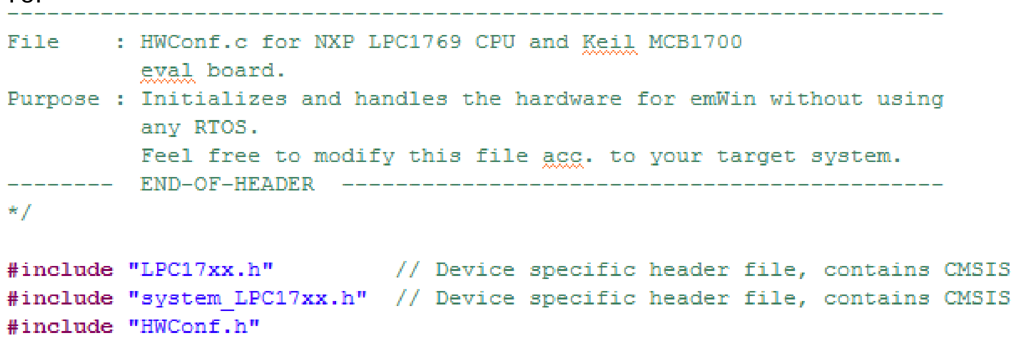
After removing all LPC1788 related files from the project, all references to the LPC1788 should be removed from the source files. A number of files need to be changed. For the MCB1700 port this means that file “System/HW/HWConf.c” must be changed from:



```
----------------------------------------------------------------------
File     : HWConf.c for NXP LPC1788 CPU and Embedded Artists LPC1788
           eval board.
Purpose  : Initializes and handles the hardware for emWin without using
           any RTOS.
           Feel free to modify this file acc. to your target system.
--------   END-OF-HEADER  ---------------------------------------------
*/

#include "LPC177x_8x.h"         // Device specific header file, contains CMSIS
#include "system_LPC177x_8x.h"  // Device specific header file, contains CMSIS
#include "HWConf.h"
```

To:



```
----------------------------------------------------------------------
File     : HWConf.c for NXP LPC1769 CPU and Keil MCB1700
           eval board.
Purpose  : Initializes and handles the hardware for emWin without using
           any RTOS.
           Feel free to modify this file acc. to your target system.
--------   END-OF-HEADER  ---------------------------------------------
*/

#include "LPC17xx.h"         // Device specific header file, contains CMSIS
#include "system_LPC17xx.h"  // Device specific header file, contains CMSIS
#include "HWConf.h"
```

## 3. Step 2: Change board-specific settings / source code

The next step is to change all code that is board-specific.

### 3.1 I$^2$C

The display board connected to the EA1788 uses an I$^2$C I/O expander, the PCA9532. The MCB1700 does not come with this IC, and the entire I$^2$C-bus remains unused. The following four more files may be removed from the project:

- System/HW/I2C_PCA9532.c
- System/HW/I2C_PCA9532.h
- System/HW/I2C.c
- System/HW/I2C.h

### 3.2 External memory

The BSP of the EA1788 uses the on-board 32 MB SDRAM memory, which the MCB1700 does not have. The LPC1788 also has an external memory bus controller while the LPC1768 does not feature this. Therefore the support for the SDRAM should be removed. In file “System/HW/HWConf.c” a number of functions must be completely removed:

- static int _TestSDRAM(void)
- static void _FindDelay(int DelayType)
- static U32 _CalibrateOsc(void)
- static void _AdjustEMCTiming(U32 Delay)
- static void _EMC_Init(void)

Some SDRAM related symbols may be removed from the “System/HW/HWConf.c” file too:

```
#define SDRAM_SIZE_MB    32
#define SDRAM_SIZE       (SDRAM_SIZE_MB * 1024 * 1024)
#define SDRAM_BASE_ADDR  0xA0000000
```

The call to function “_EMC_Init” from function “__low_level_init” in file “System/HW/HWConf.c” must be removed:

```
/*********************************************************************
*
*       __low_level_init()
*
*       Initialize clock generation and pll
*
*       Has to be modified, if another CPU clock frequency should be
*       used. This function is called during startup and
*       has to return 1 to perform segment initialization
*/
#ifdef __cplusplus
extern "C" {
#endif
int __low_level_init(void);  // Avoid "no prototype" warning
#ifdef __cplusplus
  }
#endif
int __low_level_init(void) {
  SystemCoreClockUpdate();   // Ensure, the SystemCoreClock is set
  return 1;
}
```

## 4. Step 3: Change LCD related settings / source code

The next step is to change the BSP to support the LCD used on the target hardware, for this porting example the MCB1700 board. While the EA1788 uses the on-chip LCD controller of the LPC1788, the LPC1700 does not have an on-chip LCD controller and has to rely on the LCD controller integrated in the LCD instead. When porting to another type of LCD/LCD controller, the emWin Display Driver website (http://www.segger.com/emwin-display-drivers.html) should be checked to see what driver should be used for the target hardware and how to use this driver.

### 4.1 General LCD porting hints

Each driver listed on the emWin website has its own webpage containing information on how to use the display driver. These web pages give detailed information on how to use the driver. As porting to another LCD/LCD controller heavily depends on which driver is used, a detailed description targeting all drivers cannot be given. However, there are a few general rules of thumb to follow:

- Segger's driver description webpage together with the LCD datasheet and the LCD controller datasheet should provide all required information.
- The file "Config/LCDConf.c" contains all LCD specific source code.
- When porting to another type of LCD controller, try to find an "LCDConf.c" file matching the targeted LCD controller instead of trying to change the "LCDConf.c" file from the source BSP.
- When only porting to another type of LCD while using the same LCD controller, edit the "LCDConf.c" file from the source BSP.

### 4.2 LCD porting example

The following details are specific on how the LCD specific source code and settings were ported from the EA1788 to the MCB1700.

The MCB1700 comes with a FlexColor display, using the ILI9320 LCD controller, which is supported by emWin.

#### 4.2.1 EmWin FlexColor Driver explained

Information on the FlexColor driver, which is the required driver for the MCB1700, can be found on http://www.segger.com/guidrv_flexcolor.html.

The above mentioned webpage specifies the available functions (Table 3) and which low-level functions should be provided to be able to communicate with the LCD controller (Table 4). It also states how to tell emWin to use the FlexColor driver:

```
pDevice = GUI_DEVICE_CreateAndLink(GUIDRV_FLEXCOLOR,
COLOR_CONVERSION, 0, Layer);
```

**Table 3. Available configuration functions for the FlexColor driver**

| Routine | Description |
|---|---|
| GUIDRV_FlexColor_SetFunc() | Configures bus, cache and hardware routines. |
| GUIDRV_FlexColor_Config() | Configures orientation and offset of the SEG- and COM-lines. |

Table 4. Required low-level functions when using the FlexColor API in 16-bit mode

| Element | Data type | Description |
|---|---|---|
| pfWrite16_A0 | void (*)(U16 Data) | Write single command to LCD |
| pfWrite16_A1 | void (*)(U16 Data) | Write single data word to LCD |
| pfWriteM16_A1 | void (*)(U16 * pData, int NumItems) | Write multiply data words to LCD |
| pfReadM16_A1 | void (*)(U16 * pData, int NumItems) | Read multiply data words from LCD |

**Note 1:** Besides these four functions, the user should also provide a function performing basic initialization of the display and any peripherals used to communicate with the LCD controller.

**Note 2:** The “Elements” (first column) of Table 4 are actually not functions, but pointers to functions. Eventually these pointers will each be loaded with the address where the actual function resides.

More detailed information on the FlexColor configuration functions can be found in the FlexColor driver documentation by Segger.

Though the ILI9320 supports a number of interfaces to communicate with the host, only the SPI bus can be used on the MCB1700 board. Fig 7 shows the basic transmission through SPI as specified by the datasheet of the ILI9320.

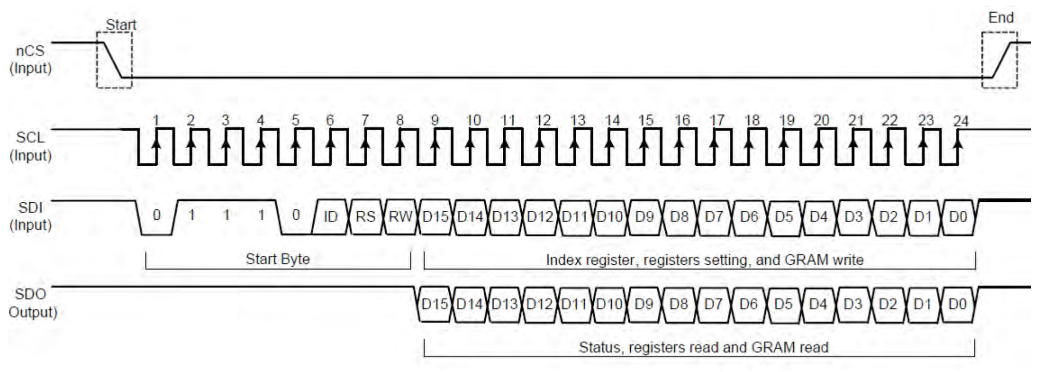

Start
End
nCS (Input)
SCL (Input)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
SDI (Input)
0 1 1 1 0 ID RS RW D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
Start Byte
Index register, registers setting, and GRAM write
SDO Output)
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
Status, registers read and GRAM read

Fig 7. Basic data transmission through SPI as specified in the ILI9320 datasheet

Segger’s API can be configured to communicate over an 8-bit bus or over a 16-bit bus. As the datasheet specifies 16-bit wide transmissions over SPI, the FlexColor API must be configured for 16 bit.

### 4.2.2 Editing the “Config/LCDConf.c” file

After analyzing the driver documentation, begin the porting of the LCD related source code by editing the “Config/LCDConf.c” file. This file contains the required source code for emWin to specify which emWin display driver should be used, how this driver communicates with the LCD controller and what type of LCD is used.

**Note:** A full listing of the “LCDConf.c” file is available in Appendix A: LCDConf.c listing.

#### 4.2.2.1 Finding a suitable “LCDConf.c” file

As mentioned in Chapter 4.1, when porting to another type of LCD controller it usually makes more sense to try to find an “LCDConf.c” file matching the targeted LCD / LCD controller instead of using the “LCDConf.c” from the source BSP as a base. Segger supplies a number of sample configuration files with emWin, which can be found in the “Sample/LCDConf/” folder. For the MCB1700 port the “Sample/LCDConf/GUIDRV_FlexColor/66708_C16_240x320/LCDConf.c” has been used. This file, together with header file “Sample/LCDConf/GUIDRV_FlexColor/LCDConf.h” are copied to the “Config/” folder, overwriting the “Config/LCDConf.c” and “Config/LCDConf.h” files.

To get this file to work on the MCB1700 the following changes are made:

##### Including a new low-level interface header file

First the inclusion of file "LCD_X_8080_16.h" is replaced with the inclusion of the “LCD_X_SPI.h” file. This file does not exist yet, but should contain low-level functions for the driver to communicate with the LCD controller. Further down this chapter the content of the “LCD_X_SPI” source- and header file are discussed.

##### Changing the resolution and orientation

The MCB1700 is designed to operate the display in landscape mode, thereby transforming the 240x320 display into a 320x240 display. Therefore, symbols “XSIZE_PHYS” and “YSIZE_PHYS” should have a value of 320 and 240, respectively. Symbol DISPLAY_ORIENTATION should be set to GUI_SWAP_XY to effectively swap the x- and y-axis.

Note: Depending on which type of MCB1700 is used, it might also be necessary to set the DISPLAY_ORIENTATION to “GUI_MIRROR_X | GUI_SWAP_XY”.

##### Color conversion

The “COLOR_CONVERSION” symbol determines how the data words are mapped to colors. The internal graphical RAM of the ILI9320 is 18-bit, while in SPI mode only 16 bits per pixel are transferred. Fig 8 shows how these 16 bits are mapped to the internal graphical RAM.

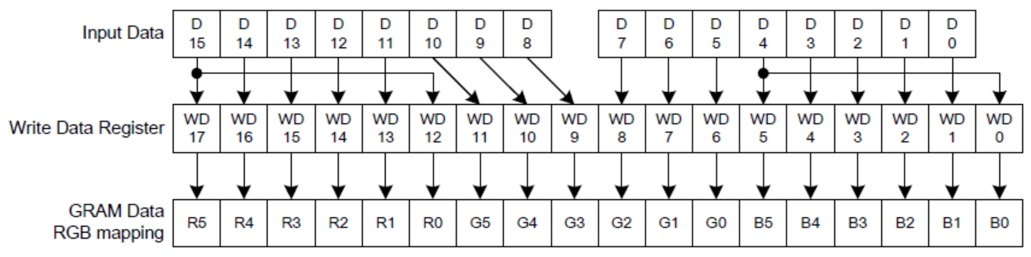


**Fig 8. Color mapping of the ILI9320 in SPI mode**

- Bits 0-4(5 bits) determine the blue color,
- Bits 5-10 (6 bits) determine the green color and
- Bits 11-15 (5 bits) determine the red color.

emWin supports a large number of color mappings which can be found in Segger's emWin user manual (chapter 13.3 "Fixed Palette Modes"). The mode corresponding with the ILI9320 format is GUICC_565, meaning that symbol "COLOR_CONVERSION" should be set to GUICC_565. Also the FlexColor driver webpage explains that a 565 format is used.

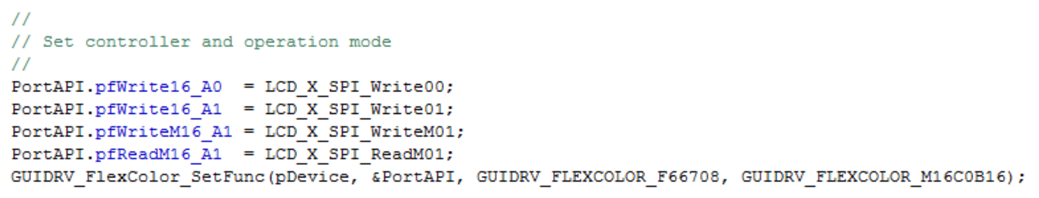
### Supplying a proper LCD controller initialization routine

After these changes only some functions should be added to allow communication to, and initialization of, the LCD controller. The configuration function from the sample "LCDConf.c" file is not suitable as this assumes the parallel bus to be used and thus must be replaced. All items related to the initialization may be removed and the function "_InitController" should be replaced with a suitable one.

Working drivers are already provided with most available LCDs by an example, which can be used as a base for writing the required low-level functions. For the MCB1700 emWin BSP an example by Keil (MCB1700 LCD_Blinky) has been used as a base, adapted for use with emWin and optimized for speed. If such an example does not exist for the targeted LCD, the required functions should be written with the aid of the documentation from Segger and the LCD's datasheet. Please see the "LCDConf.c" file (Appendix A: LCDConf.c listing or the MCB1700 BSP) for the full initialization function.

### Setting up the low-level interface pointers

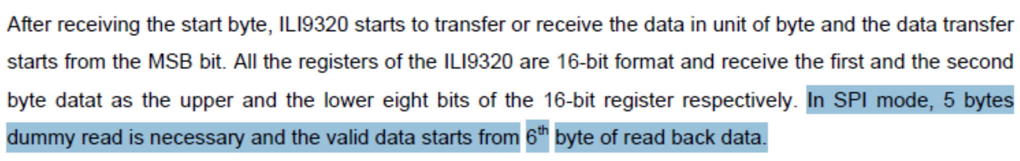
Next, the function pointers specified in Table 4 must be set in order for emWin to use the correct low-level routines to allow communication to the display controller. As previously explained in this chapter, an additional file shall contain the low-level functions which are used for communicating with the LCD controller. Fig 9 shows the loading of these function pointers.



```
//
// Set controller and operation mode
//
PortAPI.pfWrite16_A0  = LCD_X_SPI_Write00;
PortAPI.pfWrite16_A1  = LCD_X_SPI_Write01;
PortAPI.pfWriteM16_A1 = LCD_X_SPI_WriteM01;
PortAPI.pfReadM16_A1  = LCD_X_SPI_ReadM01;
GUIDRV_FlexColor_SetFunc(pDevice, &PortAPI, GUIDRV_FLEXCOLOR_F66708, GUIDRV_FLEXCOLOR_M16C0B16);
```

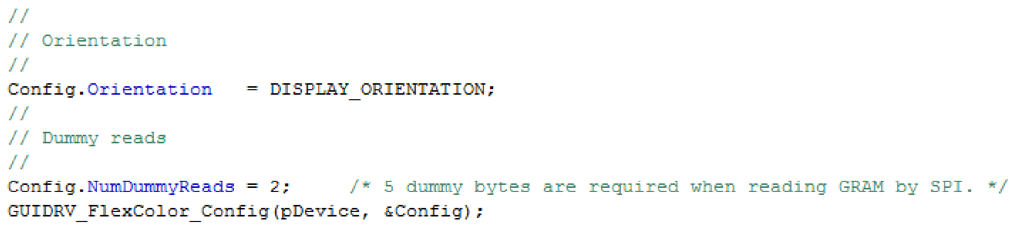
**Fig 9. Loading of FlexColor-driver function-pointers with low-level SPI communication functions**

In some rare cases it's necessary for emWin to know the content of the screen, e.g. when inverting (part of) the screen or when using an XOR draw. In the case of the MCB1700 port, the board does not have enough RAM data to hold a full frame-buffer. If no frame-buffer is used, these kinds of functions require a read from the graphical memory of the LCD (by calling the function specified by the pfReadM16_A1 pointer). The ILI9320 requires a number of dummy reads before valid content is transmitted when reading from the graphical RAM. The number of dummy reads depends on the interface used; for the parallel interface a single dummy read is enough, while the SPI interface requires five dummy reads (Fig 10). The function specified by the pfReadM16_A1 pointer should do a single byte dummy read. The remaining four dummy bytes need to be explicitly defined in the FlexColor config structure (Fig 11).



After receiving the start byte, ILI9320 starts to transfer or receive the data in unit of byte and the data transfer starts from the MSB bit. All the registers of the ILI9320 are 16-bit format and receive the first and the second byte datat as the upper and the lower eight bits of the 16-bit register respectively. In SPI mode, 5 bytes dummy read is necessary and the valid data starts from 6th byte of read back data.

**Fig 10. Quote from the ILI9320 datasheet stating 5 dummy bytes are necessary when reading from GRAM though SPI**



```
//
// Orientation
//
Config.Orientation    = DISPLAY_ORIENTATION;
//
// Dummy reads
//
Config.NumDummyReads = 2;     /* 5 dummy bytes are required when reading GRAM by SPI. */
GUIDRV_FlexColor_Config(pDevice, &Config);
```

**Fig 11. Configuring the display orientation and the number of dummy reads. 2*16-bits dummy reads in addition to 1 dummy read in low-level function equals the required 5 dummy reads**

### Cleaning up

Function “_DelayMs” in the file “System/HW/HWConf.c” is not used anymore and must be removed from the source as it is not compatible with the LPC1768.

### 4.2.3 Creating low-level functions for the emWin FlexColor driver

After changing the “LCDConf.c” file, the next step is to implement the four functions mentioned in Table 4 and Fig 9 so that the emWin FlexColor driver can communicate with the LCD controller. As these functions are hardware depended, these functions are put in a separate file in the “System/HW/” folder in the MCB1700 BSP, though they could also have been placed in the “LCDConf.c” file.

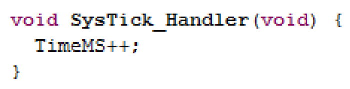
The two files that have been created to accommodate the low-level SPI functions are a new source file with accompanying header file (“LCD_X_SPI.c”, “LCD_X_SPI.h”) and are placed in the “System/HW/” folder.

Again Keil’s MCB1700 LCD_Blinky example is used as a base for these functions, adapted for use with emWin and optimized for speed. If such an example does not exist for the targeted LCD, the required functions should be written with the aid of the documentation from Segger and the LCD’s datasheet.

**Note:** A full listing of the “LCD_X_SPI.c” file is available in Appendix B: LCD_X_SPI.c listing.

### 4.2.4 Removing touch-screen source code

The EA1788 comes with touch-screen functionality, while the MCB1700 lacks this. Much of the touch-screen related source code has already been removed by replacing the “LCDConf.c” file, but there is still one place left where touch-screen related functions are referenced. In file “System/HW/HWConf.c” reference to touch-screen functionality has been removed from function “SysTick_Handler()”.



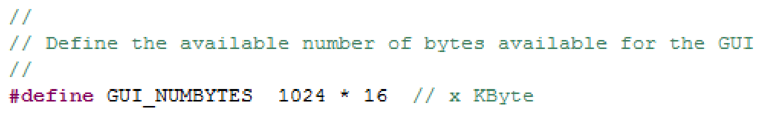
```
void SysTick_Handler(void) {
  TimeMS++;
}
```

**Fig 12. SysTick_Handler after all references to touch-screen functionality are removed**

## 5. Step 4: Change emWin related settings / source code

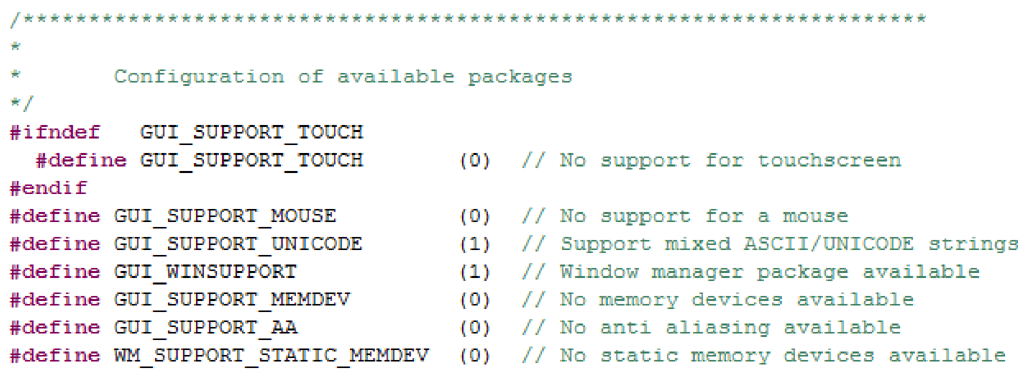
The last step is to configure emWin to be compatible with the hardware. Files "Config/GUIConf.c" and "config/GUIConf.h" are used to configure emWin.

A major difference between the EA1788 board and the MCB1700 board is the amount of on-board RAM (~32 MB vs. 64 kB). In the EA1788 BSP, emWin is given a large portion of RAM. Symbol "GUI_NUMBYTES", defined in "Config/GUIConf.c", defines how much RAM emWin is given. In the EA1788 BSP it's set to 12 MB, which should be down-sized to 16 kB to fit into the LPC1768. More information on what emWin exactly does with this allocated memory can be found in Segger's emWin user manual (http://www.segger.com/admin/uploads/productDocs/UM03001_emWin5.pdf).



```
//
// Define the available number of bytes available for the GUI
//
#define GUI_NUMBYTES  1024 * 16  // x KByte
```

File "Config/GUIConf.h" contains a number of defined symbols. The value of these symbols tailors emWin and should be set to match the hardware and provided source code. Fig 13 shows the values used for the MCB1700 BSP.



```
/*********************************************************************
*
*       Configuration of available packages
*/
#ifndef   GUI_SUPPORT_TOUCH
  #define GUI_SUPPORT_TOUCH       (0)  // No support for touchscreen
#endif
#define GUI_SUPPORT_MOUSE         (0)  // No support for a mouse
#define GUI_SUPPORT_UNICODE       (1)  // Support mixed ASCII/UNICODE strings
#define GUI_WINSUPPORT            (1)  // Window manager package available
#define GUI_SUPPORT_MEMDEV        (0)  // No memory devices available
#define GUI_SUPPORT_AA            (0)  // No anti aliasing available
#define WM_SUPPORT_STATIC_MEMDEV  (0)  // No static memory devices available
```

**Fig 13. EmWin settings as used for the MCB1700 BSP**

Support for touch-screen, mouse and memory devices are removed because the MCB1700 does not support these. Anti-aliasing is removed because of performance issues.

## 6. Overview of most important changes

Table 5. Overview of important changes

| File | Change |
|---|---|
| Config\GUIConf.c | Lowered allocated RAM for GUI from 12 MB to 16 kB. |
| Config\GUIConf.h | Re-configured emWin to match target hardware (e.g. removed touch-screen and memory device support). |
| Config\LCDConf.c | Changed LCD driver from linear to FlexColor. Changed resolution of the screen and added initialization function for the FlexColor controller. |
| System\HW\HWConf.c | Changed to support the LPC1768 instead of the LPC1788. Removed read-out of touch-screen. |
| System\HW\LCD_X_SPI.c | New source file, contains all source related to LPC1768's SSP (SPI) peripheral for communicating to the LCD controller. |
| System\HW\LCD_X_SPI.h | New header file belonging to the LPC_X_SPI.c source file. |

## 7. Reference

[1] http://www.nxp.com/redirect/segger.com/emwin.html

[2] LPC1788 Datasheet / User Manual

[3] LPC1768 Datasheet / User Manual

[4] ILI9320 Datasheet

## 8. Appendix A: LCDConf.c listing

```
/***************************************************************************
 * @file     LCDConf.c
 * @brief    Display controller configuration
 * @version  1.0
 * @date     09. May. 2012
 *
 * @note
 * Copyright (C) 2012 NXP Semiconductors(NXP), All rights reserved.
 */
#include "GUI.h"
#include "GUIDRV_FlexColor.h"
#include "LCD_X_SPI.h"

/*********************************************************************
*
*       Layer configuration (to be modified)
*
**********************************************************************
*/

//
// Physical display size
//
#define XSIZE_PHYS 320
#define YSIZE_PHYS 240

//
// Color conversion
//
#define COLOR_CONVERSION GUICC_565

//
// Display driver
//
#define DISPLAY_DRIVER GUIDRV_FLEXCOLOR

//
// Orientation
//
//#define DISPLAY_ORIENTATION (0)
//#define DISPLAY_ORIENTATION (GUI_MIRROR_X)
//#define DISPLAY_ORIENTATION (GUI_MIRROR_Y)
//#define DISPLAY_ORIENTATION (GUI_MIRROR_X | GUI_MIRROR_Y)
#define DISPLAY_ORIENTATION (GUI_SWAP_XY)
//#define DISPLAY_ORIENTATION (GUI_MIRROR_X | GUI_SWAP_XY)
//#define DISPLAY_ORIENTATION (GUI_MIRROR_Y | GUI_SWAP_XY)
//#define DISPLAY_ORIENTATION (GUI_MIRROR_X | GUI_MIRROR_Y | GUI_SWAP_XY)

/*********************************************************************
*
*       Configuration checking
*
**********************************************************************
*/
#ifndef   VXSIZE_PHYS
  #define VXSIZE_PHYS XSIZE_PHYS
#endif
#ifndef   VYSIZE_PHYS
  #define VYSIZE_PHYS YSIZE_PHYS
#endif

#ifndef   XSIZE_PHYS
  #error Physical X size of display is not defined!
#endif
#ifndef   YSIZE_PHYS
  #error Physical Y size of display is not defined!
#endif
#ifndef   COLOR_CONVERSION
```

```
  #error Color conversion not defined!
#endif
#ifndef   DISPLAY_DRIVER
  #error No display driver defined!
#endif

#define wr_reg(reg, data) LCD_X_SPI_Write00(reg); LCD_X_SPI_Write01(data);

/*********************************************************************
*
*       _InitController
*
* Purpose:
*   Initializes the LCD controller
*
*/
static void _InitController(void) {
     GUI_X_Delay(10);
     LCD_X_SPI_Init();
     GUI_X_Delay(10);

     /* Start Initial Sequence ------------------------------------------------*/
     wr_reg(0x01, 0x0100);              /* Set SS bit                         */
     wr_reg(0x02, 0x0700);              /* Set 1 line inversion               */
     wr_reg(0x04, 0x0000);              /* Resize register                    */
     wr_reg(0x08, 0x0207);              /* 2 lines front, 7 back porch        */
     wr_reg(0x09, 0x0000);              /* Set non-disp area refresh cyc ISC  */
     wr_reg(0x0A, 0x0000);              /* FMARK function                     */
     wr_reg(0x0C, 0x0000);              /* RGB interface setting              */
     wr_reg(0x0D, 0x0000);              /* Frame marker Position              */
     wr_reg(0x0F, 0x0000);              /* RGB interface polarity             */

     /* Power On sequence -----------------------------------------------------*/
     wr_reg(0x10, 0x0000);              /* Reset Power Control 1              */
     wr_reg(0x11, 0x0000);              /* Reset Power Control 2              */
     wr_reg(0x12, 0x0000);              /* Reset Power Control 3              */
     wr_reg(0x13, 0x0000);              /* Reset Power Control 4              */
     GUI_X_Delay(200);                  /* Discharge cap power voltage (200ms)*/
     wr_reg(0x10, 0x12B0);              /* SAP, BT[3:0], AP, DSTB, SLP, STB   */
     wr_reg(0x11, 0x0007);              /* DC1[2:0], DC0[2:0], VC[2:0]        */
     GUI_X_Delay(50);                   /* Delay 50 ms                        */
     wr_reg(0x12, 0x01BD);              /* VREG1OUT voltage                   */
     GUI_X_Delay(50);                   /* Delay 50 ms                        */
     wr_reg(0x13, 0x1400);              /* VDV[4:0] for VCOM amplitude        */
     wr_reg(0x29, 0x000E);              /* VCM[4:0] for VCOMH                 */
     GUI_X_Delay(50);                   /* Delay 50 ms                        */
     wr_reg(0x20, 0x0000);              /* GRAM horizontal Address            */
     wr_reg(0x21, 0x0000);              /* GRAM Vertical Address              */
     /* Adjust the Gamma Curve ------------------------------------------------*/
     wr_reg(0x30, 0x0B0D);
     wr_reg(0x31, 0x1923);
     wr_reg(0x32, 0x1C26);
     wr_reg(0x33, 0x261C);
     wr_reg(0x34, 0x2419);
     wr_reg(0x35, 0x0D0B);
     wr_reg(0x36, 0x1006);
     wr_reg(0x37, 0x0610);
     wr_reg(0x38, 0x0706);
     wr_reg(0x39, 0x0304);
     wr_reg(0x3A, 0x0E05);
     wr_reg(0x3B, 0x0E01);
     wr_reg(0x3C, 0x010E);
     wr_reg(0x3D, 0x050E);
     wr_reg(0x3E, 0x0403);
     wr_reg(0x3F, 0x0607);
     /* Set GRAM area ---------------------------------------------------------*/
     wr_reg(0x50, 0x0000);              /* Horizontal GRAM Start Address      */
     wr_reg(0x51, (XSIZE_PHYS-1));          /* Horizontal GRAM End   Address
*/
     wr_reg(0x52, 0x0000);              /* Vertical   GRAM Start Address      */
```

```
    wr_reg(0x53, (YSIZE_PHYS-1));           /* Vertical   GRAM End   Address
*/

    /* Set Gate Scan Line ---------------------------------------------------*/
    wr_reg(0x60, 0x2700);
    wr_reg(0x61, 0x0001);               /* NDL,VLE, REV                     */
    wr_reg(0x6A, 0x0000);               /* Set scrolling line               */

    /* Partial Display Control ----------------------------------------------*/
    wr_reg(0x80, 0x0000);
    wr_reg(0x81, 0x0000);
    wr_reg(0x82, 0x0000);
    wr_reg(0x83, 0x0000);
    wr_reg(0x84, 0x0000);
    wr_reg(0x85, 0x0000);

    /* Panel Control --------------------------------------------------------*/
    wr_reg(0x90, 0x0010);
    wr_reg(0x92, 0x0000);
    wr_reg(0x93, 0x0003);
    wr_reg(0x95, 0x0110);
    wr_reg(0x97, 0x0000);
    wr_reg(0x98, 0x0000);
    /* Set GRAM write direction
       I/D=11 (Horizontal : increment, Vertical : increment)                */
    /* AM=1   (address is updated in vertical writing direction)            */
    wr_reg(0x03, 0x1038);
    wr_reg(0x07, 0x0137);               /* 262K color and display ON        */
}

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/
/*********************************************************************
*
*       LCD_X_Config
*
* Purpose:
*   Called during the initialization process in order to set up the
*   display driver configuration.
*
*/
void LCD_X_Config(void) {
  GUI_DEVICE * pDevice;
  CONFIG_FLEXCOLOR Config = {0};
  GUI_PORT_API PortAPI = {0};
  //
  // Set display driver and color conversion
  //
  pDevice = GUI_DEVICE_CreateAndLink(DISPLAY_DRIVER, COLOR_CONVERSION, 0, 0);
  //
  // Display driver configuration, required for FlexColor driver
  //
  if (DISPLAY_ORIENTATION & GUI_SWAP_XY) {
    LCD_SetSizeEx (0, YSIZE_PHYS,   XSIZE_PHYS);
    LCD_SetVSizeEx(0, VYSIZE_PHYS,  VXSIZE_PHYS);
  } else {
    LCD_SetSizeEx (0, XSIZE_PHYS,   YSIZE_PHYS);
    LCD_SetVSizeEx(0, VXSIZE_PHYS,  VYSIZE_PHYS);
  }
  //
  // Orientation
  //
  Config.Orientation   = DISPLAY_ORIENTATION;
  //
  // Dummy reads
  //
  Config.NumDummyReads = 2;   /* 5 dummy bytes are required when reading GRAM by
SPI. */
  GUIDRV_FlexColor_Config(pDevice, &Config);
  //
  // Set controller and operation mode
  //
  PortAPI.pfWrite16_A0  = LCD_X_SPI_Write00;
  PortAPI.pfWrite16_A1  = LCD_X_SPI_Write01;
  PortAPI.pfWriteM16_A1 = LCD_X_SPI_WriteM01;
  PortAPI.pfReadM16_A1  = LCD_X_SPI_ReadM01;
  GUIDRV_FlexColor_SetFunc(pDevice, &PortAPI, GUIDRV_FLEXCOLOR_F66708,
GUIDRV_FLEXCOLOR_M16C0B16);
}

/*********************************************************************
*
*       LCD_X_DisplayDriver
*
* Purpose:
*   This function is called by the display driver for several purposes.
*   To support the according task the routine needs to be adapted to
*   the display controller. Please note that the commands marked with
*   'optional' are not cogently required and should only be adapted if
*   the display controller supports these features.
*
* Parameter:
*   LayerIndex - Index of layer to be configured
*   Cmd        - Please refer to the details in the switch statement below
*   pData      - Pointer to a LCD_X_DATA structure
*
* Return Value:
*   < -1 - Error
*     -1 - Command not handled
*      0 - Ok
*/
int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd, void * pData) {
  int r;
  (void) LayerIndex;
  (void) pData;

  switch (Cmd) {
  //
  // Required
  //
  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...
    //
    // ...
    _InitController();
    return 0;
  }
  default:
    r = -1;
  }
  return r;
}

/*************************** End of file ****************************/
```

## 9. Appendix B: LCD_X_SPI.c listing

```
/******************************************************************************
 * @file     LCD_X_SPI.c
 * @brief    FlexColor SPI driver
 * @version  1.0
 * @date     09. May. 2012
 *
 * @note
 * Copyright (C) 2012 NXP Semiconductors(NXP), All rights reserved.
 */

#include "GUI.h"
#include "LPC17xx.h"
#include "LCD_X_SPI.h"

/*********************** Hardware specific configuration *********************/

/* SPI Interface: SSP1

   PINS:
   - CS     = P0.6 (GPIO pin)
   - SCK    = P0.7 (SCK1)
   - SDO    = P0.8 (MISO1)
   - SDI    = P0.9 (MOSI1)                                                    */

#define PIN_CS      (1 << 6)

/*--------------- Graphic LCD interface hardware definitions -----------------*/

/* Pin CS setting to 0 or 1 */
#define LCD_CS(x)   ((x) ? (LPC_GPIO0->FIOSET = PIN_CS)    : (LPC_GPIO0->FIOCLR =
PIN_CS))

#define SPI_START   (0x70)              /* Start byte for SPI transfer */
#define SPI_RD      (0x01)              /* WR bit 1 within start */
#define SPI_WR      (0x00)              /* WR bit 0 within start */
#define SPI_DATA    (0x02)              /* RS bit 1 within start byte */
#define SPI_INDEX   (0x00)              /* RS bit 0 within start byte */

/* local functions */
__inline void wr_cmd (unsigned char cmd);                /* Write command to LCD
*/
__inline void wr_dat (unsigned short dat);               /* Write data to LCD */
__inline unsigned char spi_tran (unsigned char byte);  /* Write and read a byte
over SPI */
__inline void spi_tran_fifo (unsigned char byte);      /* Only write a byte over
SPI (faster) */

/*******************************************************************************
* Initialize SPI (SSP) peripheral at 8 databit with a bitrate of 12.5Mbps      *
*   Parameter:                                                                 *
*   Return:                                                                    *
*******************************************************************************/
void LCD_X_SPI_Init(void)
{
  uint8_t Dummy;

  /* Enable clock for SSP1, clock = CCLK / 2 */
  LPC_SC->PCONP       |= 0x00000400;
  LPC_SC->PCLKSEL0    |= 0x00200000;  /* PCLK = CCLK / 2 = 50MHz */

  /* Configure the LCD Control pins */
  LPC_PINCON->PINSEL9 &= 0xF0FFFFFF;
  LPC_GPIO4->FIODIR   |= 0x30000000;
  LPC_GPIO4->FIOSET    = 0x20000000;

  /* SSEL1 is GPIO output set to high */
  LPC_GPIO0->FIODIR   |= 0x00000040;
  LPC_GPIO0->FIOSET    = 0x00000040;
  LPC_PINCON->PINSEL0 &= 0xFFF03FFF;
  LPC_PINCON->PINSEL0 |= 0x000A8000;

  /* Enable SPI in Master Mode, CPOL=1, CPHA=1 */
  /* 12.5 MBit used for Data Transfer @ 100MHz */
  LPC_SSP1->CR0        = 0x1C7;     /* SCR = 1 */
  LPC_SSP1->CPSR       = 0x02;      /* CPSDVSR  = 2. Bit frequency = PCLK /
(CPSDVSR × [SCR+1]) = 50 / (2 × [1+1]) = 12.5Mbps */
  LPC_SSP1->CR1        = 0x02;

  while(LPC_SSP1->SR & (1<<2))
    Dummy = LPC_SSP1->DR;        /* Clear the Rx FIFO */

  LPC_GPIO4->FIOSET = 0x10000000;   /* Activate LCD backlight */
}

/*******************************************************************************
* Write command                                                                *
*   Parameter:    c: command to write                                          *
*   Return:                                                                    *
*******************************************************************************/
void LCD_X_SPI_Write00(U16 c)
{
  wr_cmd(c);
}

/*******************************************************************************
* Write data byte                                                              *
*   Parameter:    c: word to write                                             *
*   Return:                                                                    *
*******************************************************************************/
void LCD_X_SPI_Write01(U16 c)
{
  wr_dat(c);
}

/*******************************************************************************
* Write multiple data bytes                                                    *
*   Parameter:    pData:    pointer to words to write                          *
*                 NumWords: Number of words to write                           *
*   Return:                                                                    *
*******************************************************************************/
void LCD_X_SPI_WriteM01(U16 * pData, int NumWords)
{
  LCD_CS(0);
  spi_tran_fifo(SPI_START | SPI_WR | SPI_DATA);       /* Write : RS = 1, RW = 0
*/

  while(NumWords--)
  {
     spi_tran_fifo(((*pData) >>   8));           /* Write D8..D15 */
     spi_tran_fifo(((*(pData++)) & 0xFF));          /* Write D0..D7 */
  }
  while(LPC_SSP1->SR & (1<<4));                  /* wait until done */
  LCD_CS(1);
}

/*******************************************************************************
* Read multiple data bytes                                                     *
*   Parameter:    pData:    pointer to words to read                           *
*                 NumWords: Number of words to read                            *
*   Return:                                                                    *
*******************************************************************************/
void LCD_X_SPI_ReadM01(U16 * pData, int NumWords)
{
  LCD_CS(0);
  spi_tran_fifo(SPI_START | SPI_RD | SPI_DATA);       /* Read: RS = 1, RW = 1 */
  spi_tran_fifo(0);                          /* Dummy byte 1 */
  while(NumWords--)
  {
   *pData = spi_tran(0) << 8;                  /* Read D8..D15 */
   *(pDataa++) |= spi_tran(0);                /* Read D0..D7 */
  }
  while(LPC_SSP1->SR & (1<<4));                  /* wait until done */
  LCD_CS(1);
}

/*******************************************************************************
* Write a command the LCD controller                                           *
*   Parameter:    cmd:    command to be written                                *
*   Return:                                                                    *
*******************************************************************************/
__inline void wr_cmd (unsigned char cmd)
{
  LCD_CS(0);
  spi_tran_fifo(SPI_START | SPI_WR | SPI_INDEX);    /* Write : RS = 0, RW = 0 */
  spi_tran_fifo(0);
  spi_tran_fifo(cmd);
  while(LPC_SSP1->SR & (1<<4));                  /* wait until done */
  LCD_CS(1);
}

/*******************************************************************************
* Write data to the LCD controller                                             *
*   Parameter:    dat:    data to be written                                   *
*   Return:                                                                    *
*******************************************************************************/
__inline void wr_dat (unsigned short dat)
{
  LCD_CS(0);
  spi_tran_fifo(SPI_START | SPI_WR | SPI_DATA);       /* Write : RS = 1, RW = 0
*/
  spi_tran_fifo((dat >>   8));                   /* Write D8..D15 */
  spi_tran_fifo((dat & 0xFF));                   /* Write D0..D7 */
  while(LPC_SSP1->SR & (1<<4));                  /* wait until done */
  LCD_CS(1);
}

/*******************************************************************************
* Transfer 1 byte over the serial communication, wait until done and return    *
* received byte                                                                *
*   Parameter:    byte:   byte to be sent                                      *
*   Return:               byte read while sending                              *
*******************************************************************************/
__inline unsigned char spi_tran (unsigned char byte)
{
  uint8_t Dummy;

  while(LPC_SSP1->SR & (1<<4) || LPC_SSP1->SR & (1<<2))  /* while SSP1 busy or Rx
FIFO not empty ... */
     Dummy = LPC_SSP1->DR;                    /* ... read Rx FIFO */
  LPC_SSP1->DR = byte;                        /* Transmit byte */
  while (!(LPC_SSP1->SR & (1<<2)));              /* Wait until RNE set */
  return (LPC_SSP1->DR);
}

/*******************************************************************************
* Put byte in SSP1 Tx FIFO. Used for faster SPI writing                        *
*   Parameter:    byte:   byte to be sent                                      *
*   Return:                                                                    *
*******************************************************************************/
__inline void spi_tran_fifo (unsigned char byte)
{
  while (!(LPC_SSP1->SR & (1<<1)));              /* wait until TNF set */
  LPC_SSP1->DR = byte;
}

/**************************** End of file ****************************/
```

# 10. Legal information

## 10.1 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.

## 10.2 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.

## 10.3 Trademarks

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

## 11. List of figures

## 12. List of tables

## 13. Contents