LPC546XX MCUS: A 180 MHZ
ARM® CORTEX®-M4-BASED MCU
FOR YOUR COST-EFFECTIVE HMI
APPLICATIONS

BRENDON SLADE
DIRECTOR, LPC EMBEDDED ECOSYSTEM

AMF-DES-T2637 | JUNE 2017
AGENDA

• Session Objectives
• LPC portfolio and the LPC546xx overview
  - LPC ecosystem and the MCUXpresso IDE
  - Loading, Building and Running Your First SDK Example
• emWIN
  - emWIN overview and basic features
  - Hands on session with emWIN
  - Overview of other emWIN features
Session Objectives

• Understand the range of LPC MCUs that can be used for graphical applications
• Learn how to use MCUXpresso IDE to develop and debug applications
• Understand how to construct and customize a touch screen GUI using emWin
LPC Microcontroller Portfolio

Performance

Mainstream

Entry Level

Cortex-M0/M0+

Cortex-M3

Cortex-M4F

LPC1800
Performance & Integration

LPC4300
High Performance & Integration

LPC1700
Breadth & Scalability

LPC54000
Power Efficiency & Integration

LPC1200
Robust & Reliable

LPC1500
Motion Control

LPC4000
Mid-Range Performance & Integration

LPC1100
Scalable Entry Level

LPC1300
Entry Level

LPC800
8-bit Simplicity

LPC1500
Motion Control

LPC1200
Robust & Reliable

LPC800
8-bit Simplicity

LPC1100
Scalable Entry Level

LPC1300
Entry Level

LPC1200
Robust & Reliable

LPC1100
Scalable Entry Level

LPC800
8-bit Simplicity

LPC1300
Entry Level

LPC1200
Robust & Reliable

LPC1100
Scalable Entry Level

LPC800
8-bit Simplicity

LPC1300
Entry Level

LPC1200
Robust & Reliable

LPC1100
Scalable Entry Level

LPC800
8-bit Simplicity

LPC1300
Entry Level
LPC546xx Series Block Diagram

CPU
- 180MHz Cortex-M4 with floating point unit

Memory
- Up to 512 KB Flash, Up to 200 KB RAM
- 16 KB EEPROM

Interfaces for connectivity & sensors
- Dual CAN2.0 or CAN FD Controller Options
- XTAL-less FS USB (H/D)
- 10 SPI, 10 I2C, 10 UART, 2 I²S channels.
- Max 10 channels
- Graphic LCD with resolutions up to 1024x768
- 10/100 Ethernet Controller with PTP
- Stereo DMIC subsystem
  - (PDM, decimator, HW VAD)
- 1x HS USB (H/D) w/ on-chip HS PHY
- XIP from QSPI via SPIFI
- External Memory Ctrl (up to 32 bits)

Other
- Operating voltage: 1.71 to 3.6V
- Temperature range: -40 to 105 °C
- LQFP208, LQFP100
- TFBGA180, TFBGA100
Graphics LCD Controller

• Key features
  - Support for STN and TFT panels
  - **Up to 1024x768 resolution**
  - 24-bit LCD interface supports 24bpp (16M colors)
  - Palette table allows display of up to 256 of 64K colors
  - Adjustable LCD bus size supports various panel bus configurations
  - Dedicated LCD DMA controller
  - Hardware cursor support

• Graphic Library Support
  - Segger's emWin graphic library free to use with NXP’s microcontrollers
  - Other supported graphic libraries include Draupner’s TouchGFX, TARA Embedded Wizard, Expresslogic GUIX and MicroEJ

• MCUXpresso SDK support
  - Significantly reduces your software porting effort
LPC546xx Target Applications

**Industrial, Control & General Embedded**
- Industrial gateway
- HVAC control
- Building control & automation
- Diagnostic equipment
- Electronic instruments
- Multi-node comms hubs
- Multi-protocol bridge
- Various HMI/GUI apps
- Scanners
- Mini printers

**Smart Home & General Consumer**
- White Goods HMI
- Smart Small Appliance
- Thermostat
- Security monitoring & alarm
- Fitness equipment
- Audio accessories / Musical instruments

**Automotive Aftermarket**
- OBD-II
- Data collectors, Infotainment/navigation
- Telematics
- Tachograph
- Fleet Management

**Smart Energy**
- Smart Electric Meter
- In Home Display (IHD)
- Data Aggregator
- Communications Hub
- PLC, inverters, circuit breakers
LPCXpresso54608 Development Board

- LPCXpresso V3 feature set
  - Built-in CMSIS-DAP/J-link debug probe, option for using off-board probe
- LPC54608 MCU running at 180MHz
- 4.3” TFT LCD (272x480) cap touch display
- 128Mb Micron SDRAM
- 128Mb Micron quad SPI flash
- Ethernet, DMIC, SD card ports
- Stereo audio codec
- Arduino UNO R3 compatible expansion ports
- USB full and high speed ports, with Host or Device options
LPC ECOSYSTEM and MCUXPRESSO IDE
MCUXpresso Software and Tools

- IDE
- SDK
- Config Tools

For NXP’s ARM® Cortex®-M controllers
- Kinetis MCUs
- LPC Microcontrollers
- i.MX Application Processors

MCUXpresso IDE
Edit, compile, debug and optimize in an intuitive and powerful IDE

MCUXpresso SDK
Runtime software including peripheral drivers, middleware, RTOS, demos and more

MCUXpresso Config Tools
Online and desktop tool suite for system configuration and optimization
MCUXpresso Config Tools

MCUXpresso Config Tools is a suite of evaluation and configuration tools that helps guide users from first evaluation to production software development.

SDK Builder packages custom SDKs based on user selections of MCU, evaluation board, and optional software components.

Pins, Clocks, and Peripheral tools generate initialization C code for custom board support. Features validation of inputs and cross-tool conflict resolution.

Project Generator creates new SDK projects with generated Pins and Clocks source files.

Project Cloning creates a standalone SDK project based on an example application available within SDK release.

Power Estimation tool provides energy and battery-life estimates based on a user’s application model.

Available as a standalone tool for select devices.

Learn more at: www.nxp.com/mcuxpresso/config
LPCOpen

- Software drivers and libraries for pre-2017 LPC Cortex-M devices
- MCU peripheral device drivers
- Common APIs across device families
- Maintained and supported
- Includes USB host/device, Ethernet, FatFS, SWIM graphics and emWIN examples
- Keil, IAR, MCUXpresso IDE/LPCXpresso projects
- http://www.nxp.com/lpcopen

LPCOpen and Code Bundles are included in and supported by MCUXpresso IDE
GETTING STARTED
Hands-on – Session #1

• Hands-on session with following objectives:
  – Get familiar with importing SDK into MCUXpresso IDE
  – Building and running a basic example (blinky) on the LPC54608 LPCXpresso board
  – Get familiar with debugging operations
Hands-on – Session #1

- Copy MCUXpresso IDE installer executable to your PC*
- Run MCUXpresso IDE installer*

*Tools may already be installed on training laptops
Follow along: Start MCUXpresso IDE – New Workspace

• Start MCUXpresso IDE on your system
• At the dialog box, enter a location for your workspace then click OK
• Suggest C:\MCUX_om13092\workspace1
• Note: A workspace is just a folder containing the projects that you want to actively work on during this IDE session; new folders for workspace locations can be added using the Browse.. button
Follow along: New Workspace after Creation

- MCUXpresso IDE will startup in your new empty workspace with no initial projects.

- Note that Welcome Page shows that IDE has been activated (in this screenshot with a Pro Edition License).
Follow along: Develop Perspective

- A “perspective” is a collection of different “views”
- The Develop Perspective was created to provide a single combined Project Management and Debugging view
- In addition to the default Develop perspective, the MCUXpresso IDE also supports traditional Eclipse C/C++ and Debug perspectives
SDK installation process

• All LPC products released after 2016 (except LPC8xx devices) and all Kinetis products require an SDK to be installed before development can start
• SDK packages are built using the MCUXpresso Configuration Tool
• For this tutorial session, an SDK for LPCXpresso54608 has been pre-built
• The SDK includes not only a large set of drivers and examples, but also a complete set of information about the device(s) and board(s) that it supports
  - but User does not normally need to examine the SDK in detail
• Can be useful to check available drivers and example code without needing to go through the Import project steps
• **SDK installation is a simple drag and drop operation**
Follow along: SDK installation process

- Copy LPCXpresso54608 SDK zip file to your PC
- Install the SDK by dragging and dropping the zip file from a file explorer over the Installed SDK window.
- After installation of SDK is complete, click on the package in “Installed SDK” window. This shows various sections (folders) in SDK package.
- Expand each section to view the various components of that section.
Follow along: Browsing an SDK installation

Click on package open
SDK package exploration tools

Expand each section to view attributes of the SDK package loaded
Follow along: Importing an SDK example

1) Click Import SDK example(s) in Quickstart panel

2) Click on LPCXpresso54608 image to select as target board

3) Click next

SDK Import Dialog Window opens

Target MCU updates to match board MCU
Follow along: Importing an SDK example

1) Unpack / process
2) Click "Add..."
3) Select NXP example
4) Expand driver_examples and select gpio > led_output
5) Finish
6) Click to expand project content

After unpacking / processing, project appears in the project explorer window
Follow along: Building SDK example

7) Click on the project top level to make it active

8) Click Build

After selecting a project, options to Build, Clean and Debug become active

When Build starts, Console panel becomes active (bottom right of IDE) and shows build progress
LPCXpresso54608 Board Configuration

Connect USB cable from PC to “Link” port (J8) to power main board and debug probe

Before connecting to USB, ensure that “DFU Link” (JP5) jumper is fitted so that MCUXpresso IDE can soft-load probe firmware
Follow along: Start Debug Session

Select project in the Project Explorer View, then click on Debug in the Quickstart Panel.

The IDE should identify and boot the Link2 with CMSIS-DAP firmware image.

Allow Access for Windows Firewall warnings to Domain and Private networks (will only appear first time).

Note: By default, selecting “Debug” will trigger a build before the debug session is launched, so it is not necessary to run a “build” first.

Note: use Quickstart Debug, not this icon.

Click OK

These settings will be remembered for next time you debug this project.
Follow along: Develop Perspective – Debugging

Run Controls

Registers and Peripherals views

Debug view

Editor view

Variables, Global Variables and Breakpoints views

Console, Memory and Trace views
Follow along: Develop Perspective – Debugging

- Image downloaded to flash and execution started
  - Default breakpoint set on function main()
- Debug View displayed automatically
  - Shows / controls current scope and target (multicore)
  - Run controls are on main toolbar
Follow along: Develop Perspective – Debugging

- Click Resume (or press F8 key on keyboard) to execute the application.
- Press and hold switch SW2(ISP2) to make LED3 blink. Press and hold switch SW4(ISP0) to make LED1 blink.
- Click Suspend to pause execution.
- Click Restart to go back to beginning of application.
- Click Terminate to quit the debug session
emWin Overview
emWin Introduction

- Reliable embedded GUI middleware from SEGGER
- Hardware friendly
- Needs few kB of flash and <= 1kB of RAM to run core functions
- Supports all kinds of LCD panels/modules
- Comprehensive GUI features
emWIN Licensing

- No royalty or license fee with any current NXP ARM Cortex M0, M3 or M4 MCU
- Source code available under license agreement from SEGGER
- Multiple deployed customer applications using emWin on LPC:
  - ATM machines, paper money counter, Industrial touch panel, Elevator control, Security panel

  - Free to download and use from nxp.com, includes library and set of Windows utilities, GUI builder, bitmap & font converters
  - Can be used with MCUXpresso, Keil, IAR and other ARM development tools
  - Examples provided with LPCOpen (pre 2017 devices) and SDK (LPC54000 series)

emWIN is included in MCUXpresso SDK packages (for LPC546xx)
What NXP customers used emWin for...

- Paper money counter (LPC4300 + 3.1” LCD)
- ATM (LPC1800 + 14”, 1024x768 LCD)
- Industrial touch panel (LPC1788 + 10.1”, 640x480 LCD)
- Washing machine (LPC3000)
- Elevator control with LCD (LPC1788)
- High accuracy scales with LCD (LPC1788)
- Security Panel (LPC2132)
Tools overview

- **Bitmap converter** - Can be used for converting images into C-files or streamed bitmaps.

- **emWinView** - Can be used to view the content of the display while stepping through the simulation.

- **Bin2C** - Can be used for converting any kind of file into byte arrays which then can be used within a C file.

- **U2C** - Can be used to convert UTF8-text into C-code

- **Font converter** - Can be used for converting any font installed on the host system into emWin compatible formats.

- **GUIBuilder** - Can be used to generate dialog based applications without writing any line of code.
GUI-Builders

The GUI-Builders is a tool for creating dialogs without using C.

Basic usage of the GUI-Builders:

- **Check project path** in `GUIBuilder.ini`
  This file can be found in the application folder of the tool.
- **Start GUI-Builders**
- **Start with FRAMEWIN or WINDOW widget**
  Only these widgets are able to serve as parent windows for a dialog here.
- **Place widgets within the parent window**
  The widgets can be placed and sized by moving them with the mouse and/or by editing the properties in the property window.
- **Configure the widgets**
  The context menu shows the available options.
- **Save dialog**
  Each dialog is saved in a separate file. The filenames are generated automatically by the name of the parent window.

The filenames are automatically generated by the name of the parent window: `<WindowName>Dlg.c`

**Capabilities:**

- Files can be opened by drag and drop
- Multiple dialogs allowed simultaneously
- Each dialog is saved in a separate file
- Filenames are generated automatically
Skinning

- Skinning is used to change the appearance of one or multiple widgets. Currently emWin supports 2 skins:
  - **Default skin**
    Old classic style. Look can be changed by using the API functions described in the 'Widget' chapter of the documentation.
  - **FLEX_SKIN**
    Flexible skin, can easily be modified by a custom skinning routine.

- The default skin for each kind of widget can be set by:
  `<WIDGET>_SetDefaultSkin()`

- The skin of each single widget can be set by:
  `<WIDGET>_SetSkin()`

- Properties of FLEX_SKIN can be fetched / set by:
  `<WIDGET>_GetSkinFlexProps()`
  `<WIDGET>_SetSkinFlexProps()`

```c
#include "DIALOG.h"

static int _ScrollbarSkinCust(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
    switch (pDrawItemInfo->Cmd) {
    case WIDGET_ITEM_CREATE:
        WM_SetHasTrans(pDrawItemInfo->hWin);
        break;
    case WIDGET_ITEM_DRAW_BUTTON_L:
    case WIDGET_ITEM_DRAW_BUTTON_R:
        GUI_SetColor(GUI_GRAY);
        GUI_FillRoundedRect(pDrawItemInfo->x0, pDrawItemInfo->y0,
                           pDrawItemInfo->x1, pDrawItemInfo->y1, 4);
        GUI_SetColor(GUI_WHITE);
        GUI_DrawRoundedRect(pDrawItemInfo->x0, pDrawItemInfo->y0,
                             pDrawItemInfo->x1, pDrawItemInfo->y1, 4);
        GUI_SetColor(GUI_BLACK);
        GUI_FillCircle((pDrawItemInfo->x1 + pDrawItemInfo->x0) / 2,
                       (pDrawItemInfo->y1 + pDrawItemInfo->y0) / 2, 4);
        break;
    case WIDGET_ITEM_DRAW_SHAFT_L:
    case WIDGET_ITEM_DRAW_SHAFT_R:
        GUI_SetColor(GUI_WHITE);
        GUI_DrawLine(pDrawItemInfo->x0, pDrawItemInfo->y0,
                     pDrawItemInfo->x1, pDrawItemInfo->y1);
        GUI_DrawLine(pDrawItemInfo->x0, pDrawItemInfo->y1,
                     pDrawItemInfo->x1, pDrawItemInfo->y0);
        break;
    default:
        return SCROLLBAR_DrawSkinFlex(pDrawItemInfo);
    }
    return 0;
}
```
Hands-on Session 2
Hands-on – Session #2

• Hands-on session with following objectives:
  - Build and run an emWin example from the LPC54608 MCUXpresso SDK
  - Modify the example to add a logo
  - Create a simple GUI using GUIBuilder tools
Follow Along: Running out of Box example

- Open a new workspace.
- From the **Quickstart Panel** -> **Import the SDK example(s)**, select the `touch_and_draw` examples under `emwin_examples` section.
- Upon clicking finish, the example is imported into the current workspace.
- Expand the project to view the project structure. Expand every folder to view the files within the folder.
emWin Configuration

• There are two compile time configuration files
  - GUIConf.h - Configuration of available GUI features
  - LCDConf.h - Display driver configuration (obsolete)
Follow Along: Build and Run example

- In the **Quickstart Panel** click **Debug**.
- This builds the project and enters debug mode.
- Click resume or press F8 to execute the example.
- Interact with the example using touch screen, buttons etc.
- Touch and drag on the LCD screen to draw.
Change Background Bitmap Image

• Use the emWin tools to convert a bitmap to C source and compiled into the project

• In the touch_and_draw example, the background is white. We will replace the background with an image.
Follow Along: Choose a Bitmap Image

• To create a new bitmap to use as the display background, you have two choices:
  – Use the available NXP logo provided in the project folder. All instructions are written for this case, so this is the easiest path.
  – Navigate the folder emWin_Tools in the package. This folder contains various emWin tools.
Follow Along: Choose a Bitmap Image

- Open the Bitmap Conversion application “BmpCvtNXP.exe”.
- Locate the file “nxplogo_small.bmp”. Drag and drop it onto the app “BmpCvt.exe”.
- Save the file in C format. Click File -> Save As, change the type to “C bitmap file” and click Save.
Follow Along: Choose a Bitmap Image

- The “Format Specification” dialog opens. Scroll to the bottom of the list and select “**True color 24 bpp**”. Then select OK. A ‘C’ source file is created in the Tools folder.

- Copy and paste the “**nxplogo_small.c**” file into the workspace **source** folder of the project (choose “Copy” when prompted to copy or link to the file)
Follow Along: emWin API to Draw Bitmap image

- The API `GUI_DrawBitmap()` is used to draw the bitmap image. Refer to the emWin user Guide for details of the API call. This API takes 3 parameters: pointer to the bitmap to display, x position and y position.

- Copy the following lines of code and paste it inside “touch_and_draw.c” as specified:
  
  - Declare the following line with global scope.
    ```c
    extern const GUI_BITMAP bmnxplogo_small;
    ```
  
  - Paste the following as first statement inside for(;;) loop.
    ```c
    GUI_DrawBitmap(&bmnxplogo_small, 130, 0);
    ```
    
  Example:
  ```c
  for (;;)
  {
    GUI_DrawBitmap(&bmnxplogo_small, 130, 0);
    ...
    ...
  }
  ```

- Build and Debug the application. Observe the updated bitmap displayed in the background of the frame.
Creating New application using emWin GUIBuilder

• The GUIBuilder application is a tool for creating dialogs without any knowledge of the C programming language.
• Instead of writing source code the widgets can be placed and sized by drag and drop
  - Additional properties can be added per context menu. Fine tuning can be done by editing the properties of the widgets.
• The dialogs can be saved as C files which can be enhanced by adding user defined code.
• These C files with the embedded user code can be loaded and modified by the GUIBuilder
Follow Along: Creation of GUI

- **Note**: The `GUIBuilder.ini` file contains the path where the `GUIBuilder.exe` tool is located. All saved files pertaining to GUIBuilder application will be in this path. Please modify the path if required. Default path in the package is: "C:\FTF 2017 Training\emWin_Tools"
  - The ini file is created the first time GUIBuilder.exe is run
- Open the `GUIBuilder` “GUIBuilder.exe “application inside the emWin_Tools folder. Examine the capabilities of the tool.
- Create a new window by clicking on the **FrameWin** dialog icon. The default size of the window is 320 x 240. This can be changed by editing the properties of the window. Edit the properties and set xSize to 480 and ySize to 272.
Follow Along: Creation of GUI

• Create a Button by clicking the Button icon. Drag the created Button to the center of the frame.

• Click the Button and change the Button properties such as Name, xPos, yPos, xSize and ySize to match the screenshot below.

   Name = “On/off”, xPos = 170, yPos = 100, xSize = 110, ySize = 40.

• Save the GUI by clicking File -> Save. This creates the file FramewinDLG.c in the emWin_Tools folder (refer to path specified in the GUIBuilder.ini file).
Follow Along: Add FramewinDLG.c file to project workspace

- To do this either drag and drop the FramewinDLG.c file into the source folder of the project in workspace or copy the FramewinDLG.c and paste it in the source folder of the workspace.
Follow Along: Add code to create window frames

- Create a new copy of the existing “touch_and_draw.c” file in source folder. To do this copy/paste the file in the project workspace in source folder and name the new file as “touchbutton.c”.
- Exclude the “touch_and_draw.c” file from build.
- Delete or Comment all lines after GUI_Init() function.
- Copy/paste the following lines after GUI_Init().

```c
CreateFramewin();
WM_Exec();
while(1)
{
    WM_Exec();
}
```

- Build and Debug the application. When the application is executed, the GUI developed using GUIBuilder appears on the LCD.
Follow Along: Add code to Control an LED

- Edit the file "touchbutton.c" to initialize LED. Paste the following lines before `GUI_Init()`. This will initialize LED3 on the board.

  ```c
  LED3_INIT(0);
  ```

- Copy paste the following lines after `GUI_Init()`.

  ```c
  CreateFramewin();
  WM_Exec();
  while(1)
  {
    
    GUI_PID_STATE pid_state;
    if (kStatus_Success != FT5406_GetSingleTouch(&touch_handle, &touch_event, &cursorPosX, &cursorPosY))
    {
      PRINTF("error reading touch controller\r\n");
    }
    else if (touch_event != kTouch_Reserved)
    {
      pid_state.x = cursorPosY;
      pid_state.y = cursorPosX;
      pid_state.Pressed = ((touch_event == kTouch_Down) || (touch_event == kTouch_Contact));
      pid_state.Layer = 0;
      GUI_TOUCH_StoreStateEx(&pid_state);
      WM_Exec();
    }
  }
  ```
Follow Along: Add code to Control an LED

- Edit the “FramewinDLG.c” file and add the following code to the USER section, around line 22:
  
  ```
#include "board.h"
  ```

- In the file “FramewinDLG.c”, add the following code to the `_cbDialog()` function for the selected notifications “CLICKED” and “RELEASED”.

  ```
CLICKED : LED3_ON();
RELEASED: LED3_OFF();
  ```

- Build and Debug the application.
- Run the application. When the button is pressed on the LCD, LED3 switches ON. When the button is released, LED3 switches OFF.
Follow Along : Add Skinning to change GUI appearance

- Skinning allows customization of the GUI across various widgets, automatically applying the same look across all items within a frame
- Add the following lines in touchbutton.c just before calling CreateFramewin();

```c
    BUTTON_SKINFLEX_PROPS Props;

    FRAMEWIN_SetDefaultSkin(FRAMEWIN_SKIN_FLEX);
    BUTTON_SetDefaultSkin(BUTTON_SKIN_FLEX);
    WIDGET_SetDefaultEffect(WIDGET_Effect_3D);
    FRAMEWIN_SetDefaultFont(GUI_FONT_32_ASCII);
    FRAMEWIN_SetDefaultTextAlign(GUI_TA_HCENTER);

    BUTTON_GetSkinFlexProps(&Props, BUTTON_SKINFLEX_PI_FOCUSED);
    Props.aColorFrame[0] = 0x007FB13C;
    Props.aColorFrame[1] = 0x008FfF8F;
    Props.Radius = 6;
    BUTTON_SetSkinFlexProps(&Props, BUTTON_SKINFLEX_PI_FOCUSED);
```

- Add this include to the top of touchbutton.c:
  ```c
  #include “DIALOG.h”
  ```
- Build and Debug the application. Test the code by executing the application.
Other emWIN Features
emWin Library Functions

The following features are covered by the base package:

• Drawing and images
  – Software alpha blending
  – Image file support for BMP, GIF, PNG and JPEG
  – Drawing of images from non addressable media (using GetData call back)
  – Sprites and cursors (also animated)
  – Lines, polygons, rectangles, arcs and circles

• Fonts and text
  – Standard font package (ASCII and ISO 8859-1)
  – LTR (Left to Right), RTL (right to Left) and bidirectional text support
  – Support for all non anti-aliased font formats
  – Drawing of values (dec, bin, hex and float)

• Virtual screens (physical screen smaller than virtual screen)
• Touch screen support
• Multitasking support
**Widget library**

**Widget = Window + Gadget**

Currently the following widgets are supported:

- Button, Checkbox, Dropdown, Edit, Framewin, Graph, Header, Iconview, Image, Listbox, Listview, Listwheel, Menu, Multiedit, Progbar, Radio, Scrollbar, Slider, Text, Treeview

Creating a widget can be done with one line of code. There are basically 2 ways of creating a widget:

- **Direct creation**
  For each widget there exist creation functions:
  - `<WIDGET>_CreateEx()` : Creation without user data.
  - `<WIDGET>_CreateUser()` : Creation with user data.

- **Indirect creation**
  Indirect means using a dialog box creation function and a GUI_WIDGET_CREATE_INFO structure which contains a pointer to the indirect creation routine:
  - `<WIDGET>_CreateIndirect()` : Creation by dialog box creation function.

**Direct creation**

```c
void MainTask(void) {
    WM_HWIN hWin;
    GUI_Init();
    hWin = FRAMEWIN_CreateEx(10, 10, 100, 50, WM_HBKWIN, WM_CF_SHOW, 0, 0, "window", NULL);
    while (1) {
        GUI_Delay(100);
    }
}
```

**Indirect creation**

```c
#include "DIALOG.h"
static const GUI_WIDGET_CREATE_INFO _aDialogCreate[] = {
    { FRAMEWIN_CreateIndirect, "Window", 0, 10, 10, 100, 50 }
};
void MainTask(void) {
    WM_HWIN hWin;
    GUI_Init();
    hWin = GUI_CreateDialogBox(_aDialogCreate, GUI_COUNTOF(_aDialogCreate), NULL, WM_HBKWIN, 0, 0);
    while (1) {
        GUI_Delay(100);
    }
}
```
Common dialogs

- Common dialogs available:
  - Message boxes
  - Color selection
  - Exploring a file system

- **MESSAGEBOX**
  - Only one line of code required for a message box.

- **CHOOSEFILE**
  - Embedded file system explorer.
  - Simple callback mechanism used to get file data from application.
  - Ready to use sample for emFile available

- **CHOOSECOLOR**
  - Selection from an application defined array of colors.

- **CALENDAR**
  - Selection of a date of the gregorian calendar.
Memory devices

- **How do they work?**
  - Drawing operations can be passed to a memory device instead of the display to prevent display from flickering. A memory device is a hardware independent destination device for drawing operations.

- **What can they be used for?**
  - Preventing flickering
  - Container for decompressed images
  - Scaling and rotating
  - Fading operations
  - Window animations
  - Transparency effects

- **What does 'transparency' mean here?**
  - Memory devices with transparency 'know' the pixels which have been accessed. One additional bit is required per pixel for storing this information. Supported by 1, 8 and 16bpp memory devices.

- **Do memory devices support alpha blending?**
  - Yes, 32bpp memory devices support alpha blending, but not 'transparency'

```c
#include "GUI.h"

void MainTask(void) {
    GUI_MEMDEV_Handle hMem;
    GUI_RECT Rect = { 10, 10, 109, 109 };

    GUI_Init();
    hMem = GUI_MEMDEV_Create(Rect.x0, Rect.y0, Rect.x1 - Rect.x0 + 1, Rect.y1 - Rect.y0 + 1);
    GUI_DrawGradientH(Rect.x0, Rect.y0, Rect.x1, Rect.y1, GUI_RED, GUI_BLUE);
    GUI_MEMDEV_Select(hMem);
    GUI_DrawRectEx(&Rect);
    GUI_SetTextMode(GUI_TM_TRANS);
    GUI_DispStringInRect("This shows how transparency can be used with memory devices", &Rect, GUI_TA_HCENTER | GUI_TA_VCENTER);
    GUI_MEMDEV_Select(0);
    GUI_MEMDEV_Write(hMem);
    while (1) {
        GUI_Delay(100);
        GUI_MEMDEV_Write(hMem);
    }
}
```

![Without transparency](image1.png) ![With transparency](image2.png)
Antialiasing

- Antialiasing smoothes curves and diagonal lines by "blending" the background color with that of the foreground.

- emWin supports antialiased drawing of
  - **Text**
    Font converter is required for creating AA fonts.
  - **Arcs**
    GUI_AA_DrawArc()
  - **Circles**
    GUI_AA_FillCircle()
  - **Lines**
    GUI_AA_DrawLine()
  - **Polygons**
    GUI_AA_DrawPolyOutline()
    GUI_AA_FillPolygon()

**Note:** Performance of antialiased drawing operations degrades significantly in comparison to non antialiased drawing operations.

```c
#include "GUI.h"

static GUI_POINT _aPoint[] = {
    { -5, -5 }, { 0, -50 }, { 5, -5 }, { 50, 0 },
    { 5, 5 }, { 0, 50 }, {-5, 5 }, { -50, 0 },
};

void MainTask(void) {
    GUI_Init();
    GUI_SetBkColor(GUI_WHITE);
    GUI_SetColor(GUI_BLACK);
    GUI_Clear();
    GUI_SetPenSize(2);
    GUI_AA_DrawLine(10, 10, 100, 50);
    GUI_AA_DrawArc(100, 50, 40, 40, 270, 450);
    GUI_AA_FillCircle(50, 100, 30);
    GUI_AA.DrawPolyOutline(_aPoint, GUI_COUNTOF(_aPoint), 4, 200, 100);
    GUI_AA.FillPolygon(_aPoint, GUI_COUNTOF(_aPoint), 100, 170);
    while (1) {
        GUI_Delay(100);
    }
}
```
Window Manager

- Management system for a hierarchic window structure
  Each layer has its own desktop window. Each desktop window can have its own hierarchical tree of child windows.

- Callback mechanism based system
  Communication is based on an event driven callback mechanism. All drawing operations should be done within the WM_PAINT event.

- Foundation of widget library
  All widgets are based on the functions of the WM.

Basic capabilities:
- Automatic clipping
- Automatic use of multiple buffers
- Automatic use of memory devices
- Automatic use of display driver cache
- Motion support

```c
#include "WM.h"
void _cbWin(WM_MESSAGE *pMsg) {
  int xSize, ySize;
  switch (pMsg->MsgId) {
    case WM_PAINT:
      xSize = WM_GetWindowSizeX(pMsg->hWin);
      ySize = WM_GetWindowSizeY(pMsg->hWin);
      GUI_Clear();
      GUI_DrawRect(0, 0, xSize - 1, ySize - 1);
      GUI_DispStringHCenterAt("Window", xSize / 2, 10);
      break;
    default:
      WM_DefaultProc(pMsg);
      break;
  }
}

void _cbBk(WM_MESSAGE *pMsg) {
  switch (pMsg->MsgId) {
    case WM_PAINT:
      GUI_DrawGradientV(0, 0, 479, 271, GUI_BLUE, GUI_MAGENTA);
      break;
    default:
      WM_DefaultProc(pMsg);
      break;
  }
}

void MainTask(void) {
  WM_HWIN hWin;
  GUI_Init();
  WM_SetCallback(WM_HBKWIN, _cbBk);
  hWin = WM_CreateWindowAsChild(10, 10, 100, 100, WM_HBKWIN, WM_CF_SHOW, _cbWin, 0);
  while (1) {
    GUI_Delay(100);
  }
}
```
Supplemental Information
Design Considerations
Choosing a LCD: Resolution and Color Depth

• Resolution is not measured in inches!
• LCD resolution is quoted as **width x height** with units in pixels.
  - QVGA 320 X 240
  - VGA 640 x 480
  - SVGA 800 X 600
  - Landscape or portrait orientation
• Color depth or bits per pixel (bpp)
What is a Frame Buffer?

- **Contiguous** memory buffer containing a complete frame of data
- Consists of color values for every pixel
- Color values are commonly represented as
  - 1 bit (1 bpp): Monochrome
  - 2 bit (2 bpp): Palette based (4 colors)
  - 4 bit (4 bpp): Palette (16 colors, controller has a palette look-up table)
  - 8 bit (8 bpp): Palette (256 colors, controller has a palette look-up table)
  - 16 bit (16 bpp): High color format (5:5:5 - 32,768 colors; 5:6:5 - 65,536 colors)
  - 24 bit (24 bpp): True color format (16,777,216 colors)
Resolution x Color Depth = Memory Size

- Resolution x Color Depth = Total bits needed (divide by 8 for bytes)
- Framebuffer = Memory buffer containing complete frame (bitmap) of data

<table>
<thead>
<tr>
<th>Resolution</th>
<th>1 bits/pixel</th>
<th>2 bits/pixel</th>
<th>4 bits/pixel</th>
<th>8 bits/pixel</th>
<th>16 bits/pixel</th>
<th>24 bits/pixel</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGA</td>
<td>1024x768</td>
<td>98,304</td>
<td>196,608</td>
<td>393,216</td>
<td>786,432</td>
<td>1,572,864</td>
</tr>
<tr>
<td>WVGA</td>
<td>800x480</td>
<td>48,000</td>
<td>96,000</td>
<td>192,000</td>
<td>384,000</td>
<td>768,000</td>
</tr>
<tr>
<td>VGA</td>
<td>640x480</td>
<td>38,400</td>
<td>76,800</td>
<td>153,600</td>
<td>307,200</td>
<td>614,400</td>
</tr>
<tr>
<td>WQVGA</td>
<td>480x272</td>
<td>16,320</td>
<td>32,640</td>
<td>65,280</td>
<td>130,560</td>
<td>261,120</td>
</tr>
<tr>
<td>QVGA</td>
<td>320x240</td>
<td>9,600</td>
<td>19,200</td>
<td>38,400</td>
<td>76,800</td>
<td>153,600</td>
</tr>
<tr>
<td>CGA</td>
<td>320x200</td>
<td>8,000</td>
<td>16,000</td>
<td>32,000</td>
<td>64,000</td>
<td>128,000</td>
</tr>
</tbody>
</table>

Example: \((480 \times 272) \times 16\text{bpp} = 2,088,960\text{ bits}\)
To convert to bytes: \(2088960 \div 8 = 261,120\text{ bytes}\)
Palette Based Frame Buffer

- The frame buffer will contain an index value for each pixel
- Palette RAM is pre-filled with 16-bit color value for each index

- NXP microcontrollers have 256 entries to support
  - 1, 2, 4, or 8 bpp palletized color displays for color STN and TFT
  - 1, 2, or 4 bits-per-pixel (bpp) palletized displays for mono STN
With a 18-bit TFT display, a clock of data will drive 1 pixel. It will take 320 clocks to drive all the data for a 320 pixel line.
Driving a clocked LCD bus
Refresh Rate

- \( \text{Refresh Rate}(\text{Hz}) = \frac{\text{pixel\_clock\_rate}}{([\text{vertical\_resolution} + \text{vertical\_front\_porch} + \text{vertical\_back\_porch}] \times (\text{pixel\_clocks\_per\_data\_line} + \text{horizontal\_front\_porch} + \text{horizontal\_back\_porch})} \)

- Example for a 320 x 240 LCD display:
  - 6.5MHz pixel clock
  - vertical resolution=240 lines,
  - vertical front porch=5 lines,
  - vertical back porch=1 line,
  - pixel clocks per data line = 320 pixels,
  - horizontal front porch=20 clocks,
  - horizontal back porch=10 clocks

- \( \text{Refresh Rate} = \frac{6500000}{[(240+5+1)\times(320+20+10)]} = 75.5 \text{ Hz} \)
LCD Controller Signals

- The largest configuration for the LCD controller uses 31 pins. There are many variants using as few as 10 pins for a monochrome STN panel.

<table>
<thead>
<tr>
<th>Pin name</th>
<th>Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCD_PWR</td>
<td>output</td>
<td>LCD panel power enable.</td>
</tr>
<tr>
<td>LCD_DCLK</td>
<td>output</td>
<td>LCD panel clock.</td>
</tr>
<tr>
<td>LCD_AC</td>
<td>output</td>
<td>STN AC bias drive or TFT data enable output.</td>
</tr>
<tr>
<td>LCD_FP</td>
<td>output</td>
<td>Frame pulse (STN). Vertical synchronization pulse (TFT)</td>
</tr>
<tr>
<td>LCD_LE</td>
<td>output</td>
<td>Line end signal</td>
</tr>
<tr>
<td>LCD_LP</td>
<td>output</td>
<td>Line synchronization pulse (STN). Horizontal synchronization pulse (TFT)</td>
</tr>
<tr>
<td>LCD_VD[23:0]</td>
<td>output</td>
<td>LCD panel data. Bits used depend on the panel configuration.</td>
</tr>
<tr>
<td>LCD_CLKIN</td>
<td>input</td>
<td>Optional clock input. Each level on this pin must be at least 1 PCLK in duration in order to be sampled. The maximum frequency must therefore be less than PCLK/2.</td>
</tr>
</tbody>
</table>
## LCD TFT Signals

<table>
<thead>
<tr>
<th>Pin name</th>
<th>12-bit, 4:4:4 mode (18 pins)</th>
<th>16-bit, 5:6:5 mode (22 pins)</th>
<th>16-bit, 1:5:5:5 mode (24 pins)</th>
<th>24-bit (30 pins)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCD_PWR</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>LCD_DCLK</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>LCD_AC</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>LCD_FP</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>LCD_LE</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>LCD_LP</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>LCD_VD[1:0]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>RED[1:0]</td>
</tr>
<tr>
<td>LCD_VD[2]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Intensity</td>
</tr>
<tr>
<td>LCD_VD[9:8]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>GREEN[1:0]</td>
</tr>
<tr>
<td>LCD_VD[17:16]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>BLUE[1:0]</td>
</tr>
<tr>
<td>LCD_VD[18]</td>
<td>-</td>
<td>-</td>
<td>Intensity</td>
<td>BLUE[2]</td>
</tr>
</tbody>
</table>
Driving the LCD – Various Timings

VSYNC starts the frame

Vertical back porch timing

Horizontal front porch timing

HSYNC starts at the beginning of each line

Vertical front porch timing

Horizontal back porch timing

320x240 display shown with timing for VSYNC, HSYNC, clock, and porch values
Snapshot of incorrect LCD settings

Picture is an example for Truly G240320LTSW panel, which wrongly sets vertical back porch to 8 while the correct value is 4, leading to 4 visible blank lines by LCD panel.
Example: 480 x 272 LCD TFT Display Timing Table

- Below is the timing table of LCD used in LPCXpresso54608 Development board.

<table>
<thead>
<tr>
<th>Item</th>
<th>Symbol</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCLK Frequency</td>
<td>Fclk</td>
<td>5</td>
<td>9</td>
<td>12</td>
<td>MHz</td>
</tr>
<tr>
<td>DCLK Period</td>
<td>Tclk</td>
<td>83</td>
<td>110</td>
<td>200</td>
<td>ns</td>
</tr>
<tr>
<td>Hsync</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Period Time</td>
<td>Th</td>
<td>490</td>
<td>531</td>
<td>605</td>
<td>DCLK</td>
</tr>
<tr>
<td>Display Period</td>
<td>Thdisp</td>
<td>480</td>
<td></td>
<td></td>
<td>DCLK</td>
</tr>
<tr>
<td>Back Porch</td>
<td>Thbp</td>
<td>8</td>
<td>43</td>
<td></td>
<td>DCLK</td>
</tr>
<tr>
<td>Front Porch</td>
<td>Thfp</td>
<td>2</td>
<td>8</td>
<td></td>
<td>DCLK</td>
</tr>
<tr>
<td>Pulse Width</td>
<td>Thw</td>
<td>1</td>
<td></td>
<td></td>
<td>DCLK</td>
</tr>
<tr>
<td>Vsync</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Period Time</td>
<td>Tv</td>
<td>275</td>
<td>288</td>
<td>335</td>
<td>H</td>
</tr>
<tr>
<td>Display Period</td>
<td>Tvdisp</td>
<td>272</td>
<td></td>
<td></td>
<td>H</td>
</tr>
<tr>
<td>Back Porch</td>
<td>Tvbpc</td>
<td>2</td>
<td>12</td>
<td></td>
<td>H</td>
</tr>
<tr>
<td>Front Porch</td>
<td>Tvfpc</td>
<td>1</td>
<td>4</td>
<td></td>
<td>H</td>
</tr>
<tr>
<td>Pulse Width</td>
<td>Tvwc</td>
<td>1</td>
<td>10</td>
<td></td>
<td>H</td>
</tr>
</tbody>
</table>
LCD Tearing

- Tearing:
  - Result of LCD DMA unable to service the LCD FIFO in time
  - Use the FIFO Underflow to monitor for this
  - Workarounds
    - Change AHB priority – next slide
    - Slow down frame refresh rate, pixel clock if possible
    - Use 32-bit wide external memories
    - Increase the SDRAM clock speed, use faster SRAM
    - Profile code and move frequently accessed code to internal SRAM
LPC54608 LCD AHB Priority

- AHB Matrix Priority Register (AHBMATPRIO - 0x4000 0010)
- The values used for the various priorities are 3 = highest, 0 = lowest

<table>
<thead>
<tr>
<th>Bit</th>
<th>Symbol</th>
<th>Description</th>
<th>Reset value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>PRIICODE</td>
<td>I-Code bus priority. Should typically be lower than PRI_DCODE for best operation.</td>
<td>0</td>
</tr>
<tr>
<td>3:2</td>
<td>PRI_DCODE</td>
<td>D-Code bus priority.</td>
<td>0</td>
</tr>
<tr>
<td>5:4</td>
<td>PRI_SYS</td>
<td>System bus priority.</td>
<td>0</td>
</tr>
<tr>
<td>7:6</td>
<td>PRI_DMA</td>
<td>DMA controller priority.</td>
<td>0</td>
</tr>
<tr>
<td>9:8</td>
<td>PRI_ETH</td>
<td>Ethernet DMA priority.</td>
<td>0</td>
</tr>
<tr>
<td>11:10</td>
<td>PRI_LCD</td>
<td>LCD DMA priority.</td>
<td>0</td>
</tr>
<tr>
<td>13:12</td>
<td>PRI_USB0</td>
<td>USB0 DMA priority.</td>
<td>0</td>
</tr>
<tr>
<td>15:14</td>
<td>PRI_USB1</td>
<td>USB1 DMA priority.</td>
<td>0</td>
</tr>
<tr>
<td>17:16</td>
<td>PRI_SDI0</td>
<td>SDIO priority.</td>
<td>0</td>
</tr>
<tr>
<td>19:18</td>
<td>PRI_MCAN1</td>
<td>MCAN1 priority.</td>
<td>0</td>
</tr>
<tr>
<td>21:20</td>
<td>PRI_MCAN2</td>
<td>MCAN2 priority.</td>
<td>0</td>
</tr>
<tr>
<td>31:22</td>
<td>-</td>
<td>Reserved. Read value is undefined, only zero should be written.</td>
<td>-</td>
</tr>
</tbody>
</table>

- To give priority to the LCD DMA use the value 0x0000 0C09
## Bus Bandwidth Calculator

<table>
<thead>
<tr>
<th>EMC Bus Clock (MHz):</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic External Memory Configuration -</td>
<td></td>
</tr>
<tr>
<td>Precharge Command Period, tRP:</td>
<td>3</td>
</tr>
<tr>
<td>RAS Latency (Active to Read/Write Delay), RAS (tRCD):</td>
<td>3</td>
</tr>
<tr>
<td>CAS Latency, CAS:</td>
<td>3</td>
</tr>
<tr>
<td><strong>LCD parameters</strong></td>
<td></td>
</tr>
<tr>
<td>Horizontal (Pixels):</td>
<td>480</td>
</tr>
<tr>
<td>Vertical (Pixels):</td>
<td>272</td>
</tr>
<tr>
<td>Horizontal back porch (pixel clocks):</td>
<td>32</td>
</tr>
<tr>
<td>Horizontal front porch (pixel clocks):</td>
<td>8</td>
</tr>
<tr>
<td>Vertical back porch (lines):</td>
<td>12</td>
</tr>
<tr>
<td>Vertical front porch (lines):</td>
<td>4</td>
</tr>
<tr>
<td>Pixel clock rate (MHz):</td>
<td>9</td>
</tr>
<tr>
<td>Color Depth (bpp):</td>
<td>16</td>
</tr>
<tr>
<td><strong>Results</strong></td>
<td></td>
</tr>
<tr>
<td>Refresh Rate (Hz):</td>
<td>60.1</td>
</tr>
<tr>
<td>Frame Buffer (KB):</td>
<td>255</td>
</tr>
<tr>
<td>LCD Data Rate (Mpixels/s):</td>
<td>7.8</td>
</tr>
<tr>
<td>LCD Data Rate (MWords/s):</td>
<td>3.9</td>
</tr>
<tr>
<td>LCD Data Rate (Mbursts/s):</td>
<td>1.0</td>
</tr>
<tr>
<td>Dynamic External Memory Burst -Burst (clocks):</td>
<td>22</td>
</tr>
<tr>
<td><strong>Bus bandwidth required (%)</strong>:</td>
<td>22.5%</td>
</tr>
</tbody>
</table>