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

SECURE CONNECTIONS FOR A SMARTER WORLD

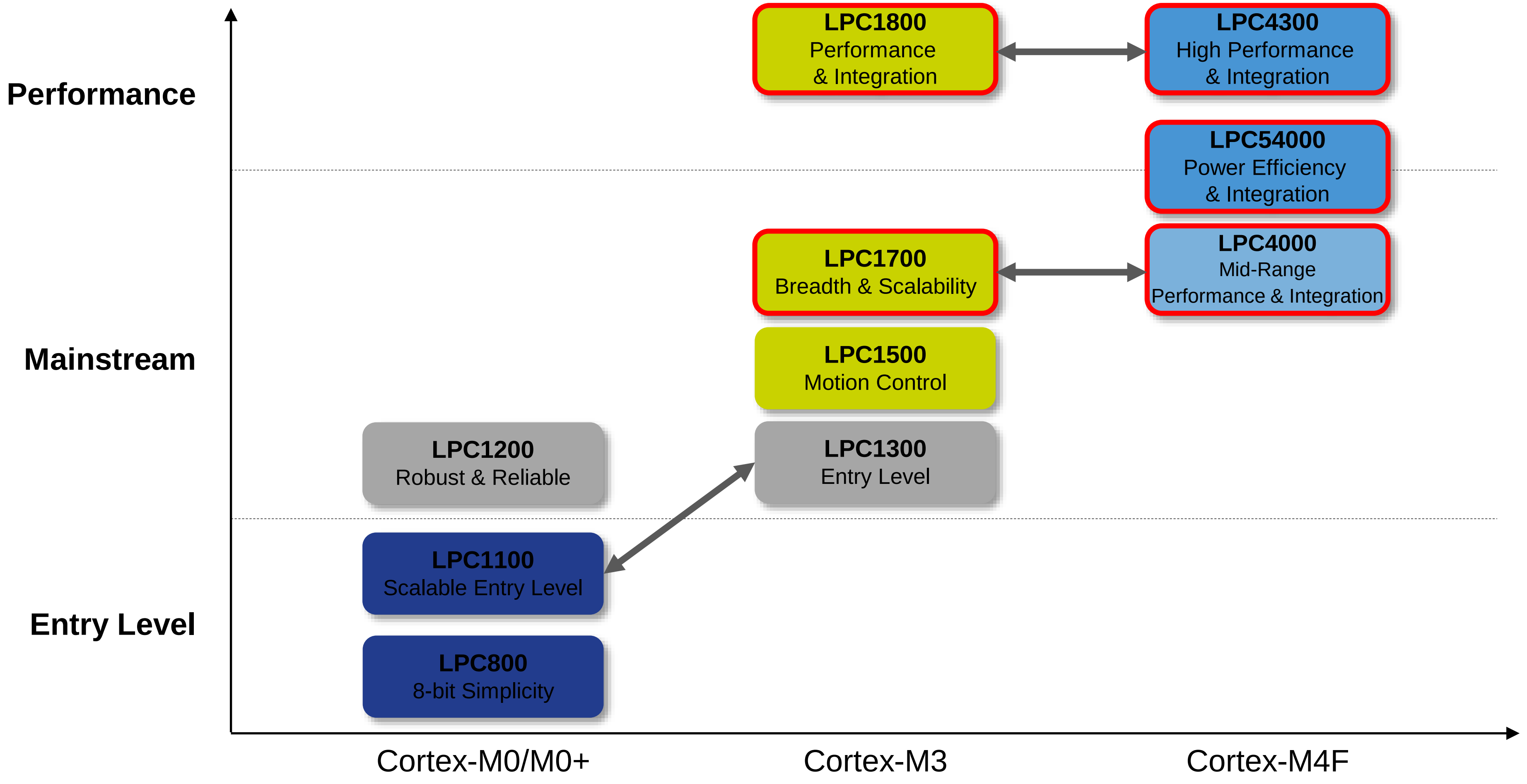
AGENDA

- 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

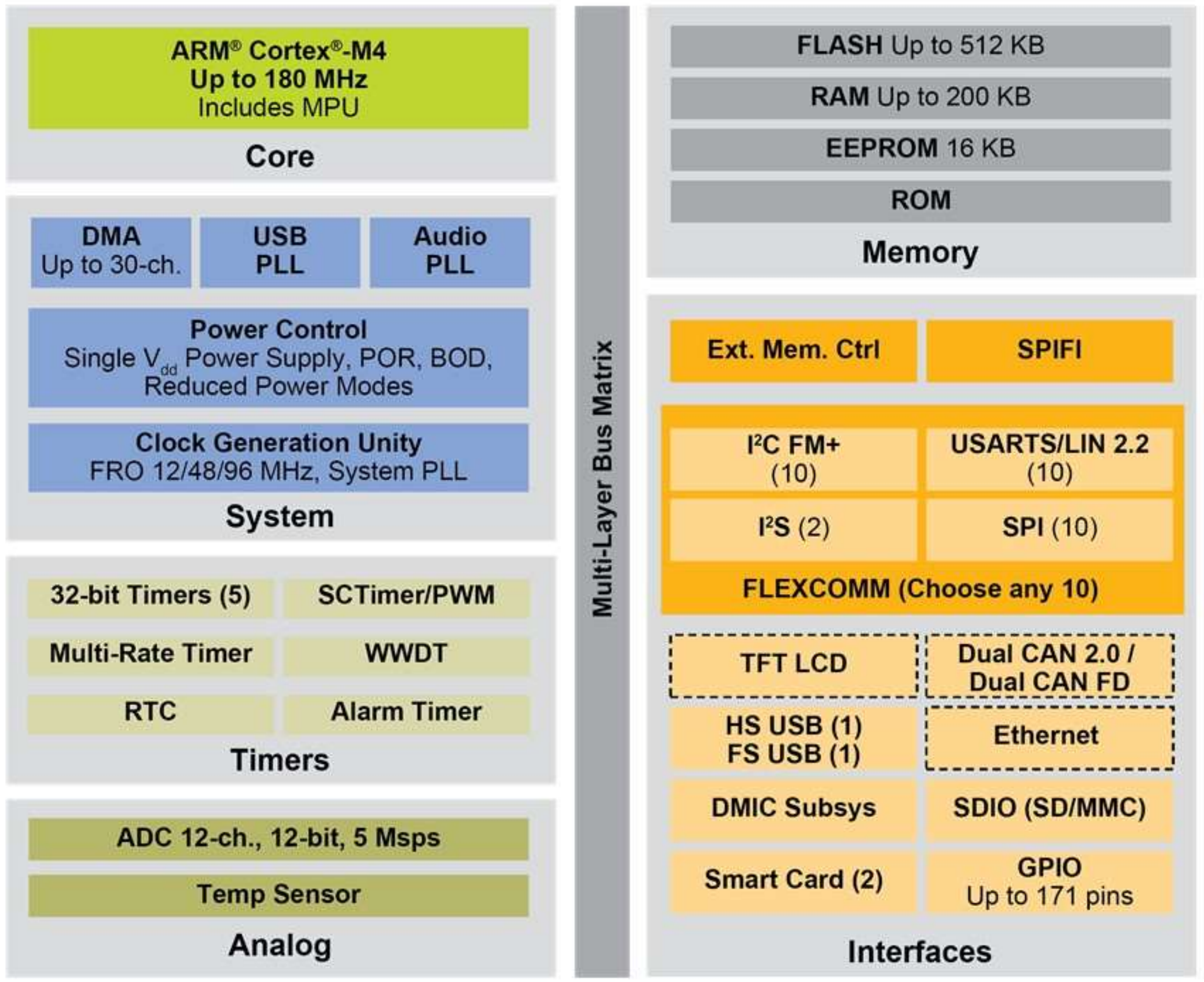
- Understand the range of LPC MCUs that can be used for graphical applications
- Learn how to use MCUXpresso IDE to develop and debug applications
- Understand how to construct and customize a touch screen GUI using emWin

LPC Microcontroller Portfolio



Performance
Mainstream
Entry Level
LPC1800 Performance & Integration
LPC4300 High Performance & Integration
LPC54000 Power Efficiency & Integration
LPC1700 Breadth & Scalability
LPC4000 Mid-Range Performance & Integration
LPC1500 Motion Control
LPC1200 Robust & Reliable
LPC1300 Entry Level
LPC1100 Scalable Entry Level
LPC800 8-bit Simplicity
Cortex-M0/M0+
Cortex-M3
Cortex-M4F

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 I^{2}S 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 & Tools

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

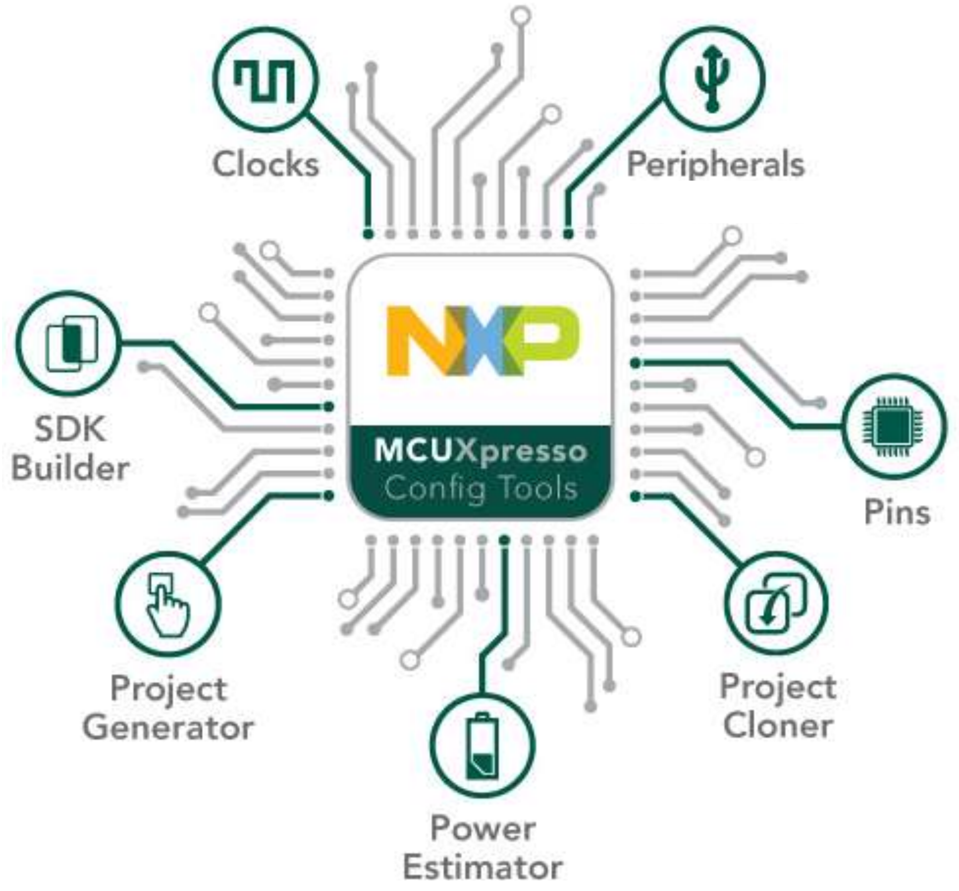
MCUXpresso Config Tools

Online and desktop tool suite for system configuration and optimization

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

MCUXpresso Config Tools

Integrated configuration and development tools for LPC and Kinetis MCUs



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 a 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.

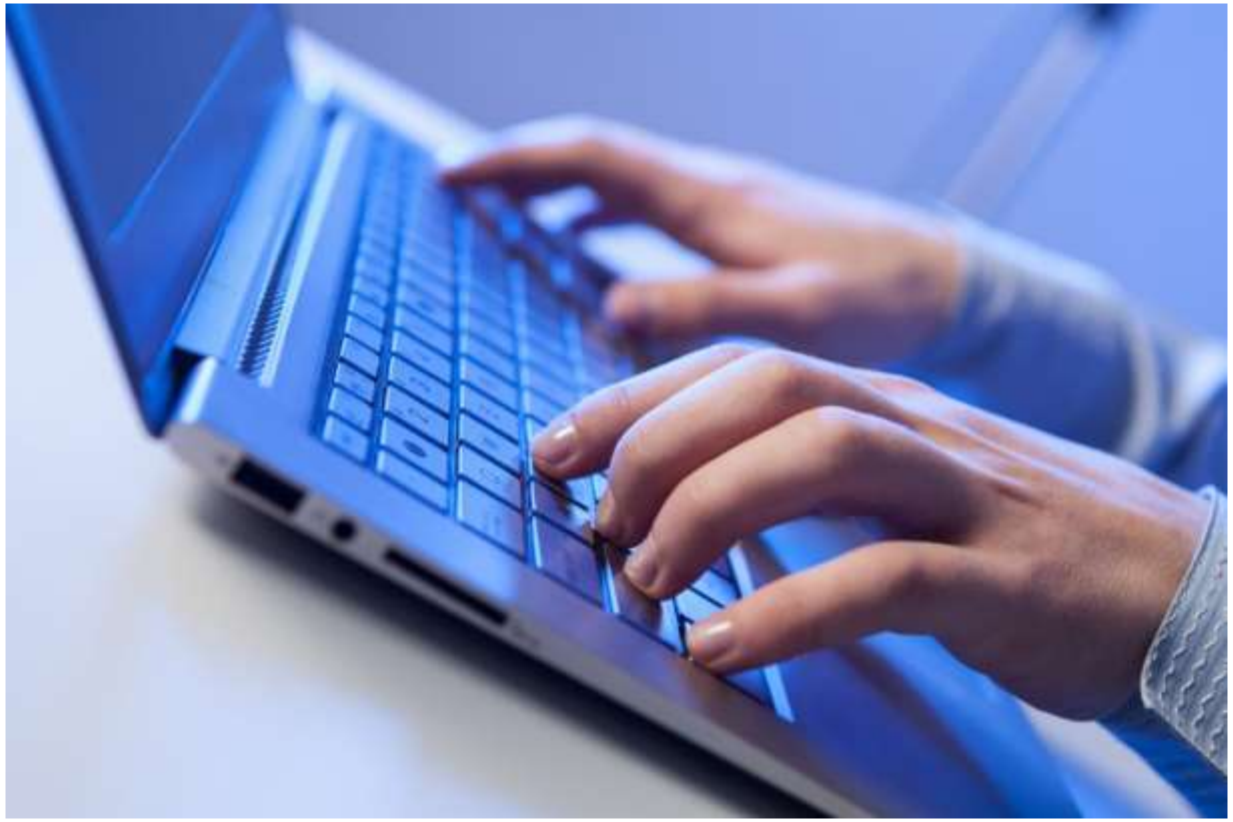
LPCOpen

- 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

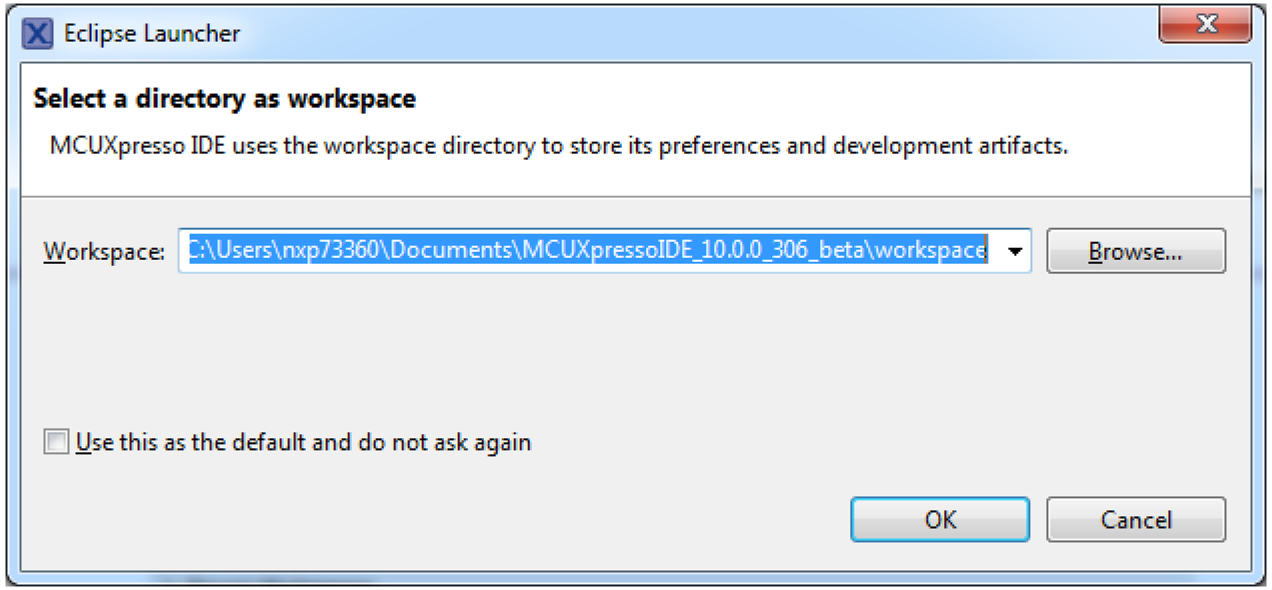
Hands-on – Session #1

- 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



Select a directory as workspace

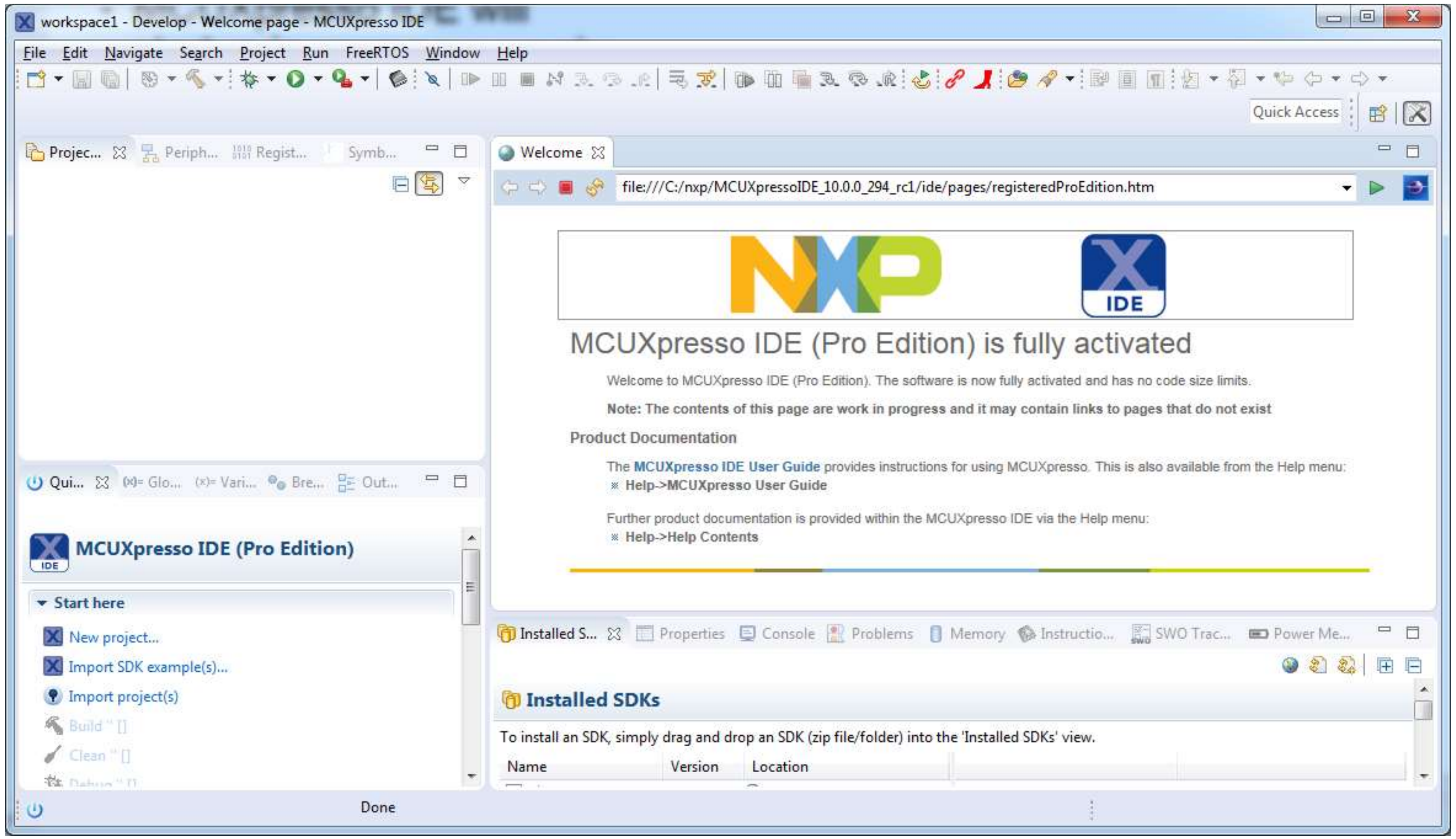
MCUXpresso IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Users\nxp73360\Documents\MCUXpressoIDE_10.0.0_306_beta\workspace

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)


workspace1 - Develop - Welcome page - MCUXpresso IDE
File Edit Navigate Search Project Run FreeRTOS Window Help
Quick Access
Projec... Periph... Regist... Symb...
Welcome
file:///C:/nxp/MCUXpressoIDE_10.0.0_294_rc1/ide/pages/registeredProEdition.htm
Qui... Glo... Vari... Bre... Out...
MCUXpresso IDE (Pro Edition)
Start here
New project...
Import SDK example(s)...
Import project(s)
Build
Clean
Installed S... Properties Console Problems Memory Instructio... SWO Trac... Power Me...
Done

MCUXpresso IDE (Pro Edition) is fully activated

Welcome to MCUXpresso IDE (Pro Edition). The software is now fully activated and has no code size limits.

Note: The contents of this page are work in progress and it may contain links to pages that do not exist

Product Documentation

The MCUXpresso IDE User Guide provides instructions for using MCUXpresso. This is also available from the Help menu:
- Help->MCUXpresso User Guide

Further product documentation is provided within the MCUXpresso IDE via the Help menu:
- Help->Help Contents

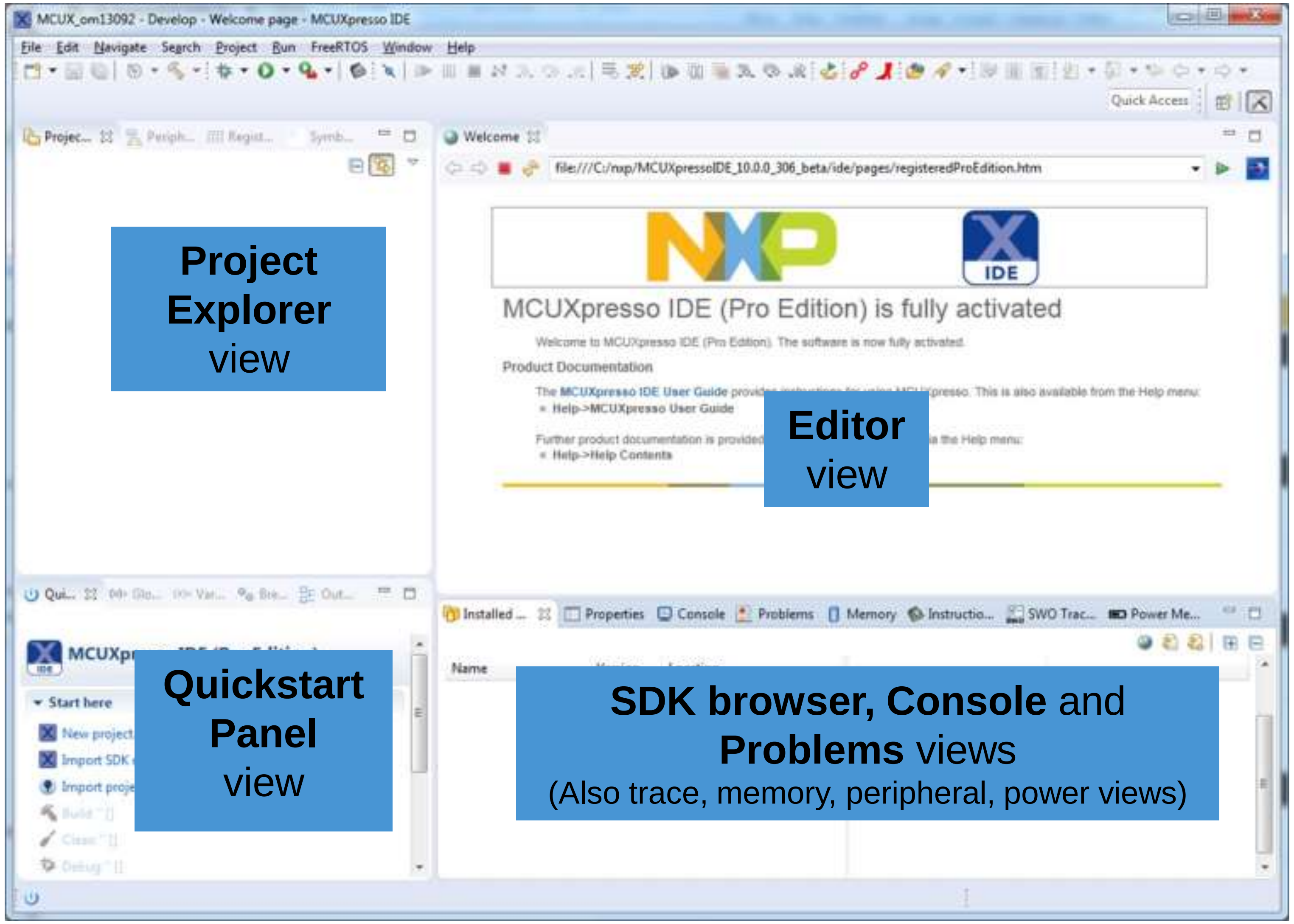
Installed SDKs

To install an SDK, simply drag and drop an SDK (zip file/folder) into the 'Installed SDKs' view.

Name	Version	Location

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



MCUX_om13092 - Develop - Welcome page - MCUXpresso IDE
File Edit Navigate Search Project Run FreeRTOS Window Help
Quick Access
Projec... Periph... Regist... Symb...
Welcome
file:///C:/nxp/MCUXpressoIDE_10.0.0_306_beta/ide/pages/registeredProEdition.htm
IDE
Project Explorer view
Editor view
Qui... Glo... Var... Bre... Out...
MCUXpr
Start here
New project
Import SDK
Import proje
Build
Clean
Debug
Quickstart Panel view
Installed ... Properties Console Problems Memory Instructio... SWO Trac... Power Me...
Name
SDK browser, Console and Problems views
(Also trace, memory, peripheral, power views)

MCUXpresso IDE (Pro Edition) is fully activated

Welcome to MCUXpresso IDE (Pro Edition). The software is now fully activated.

Product Documentation

The MCUXpresso IDE User Guide provid... MCUXpresso. This is also available from the Help menu:
- Help->MCUXpresso User Guide

Further product documentation is provided ... the Help menu:
- Help->Help Contents

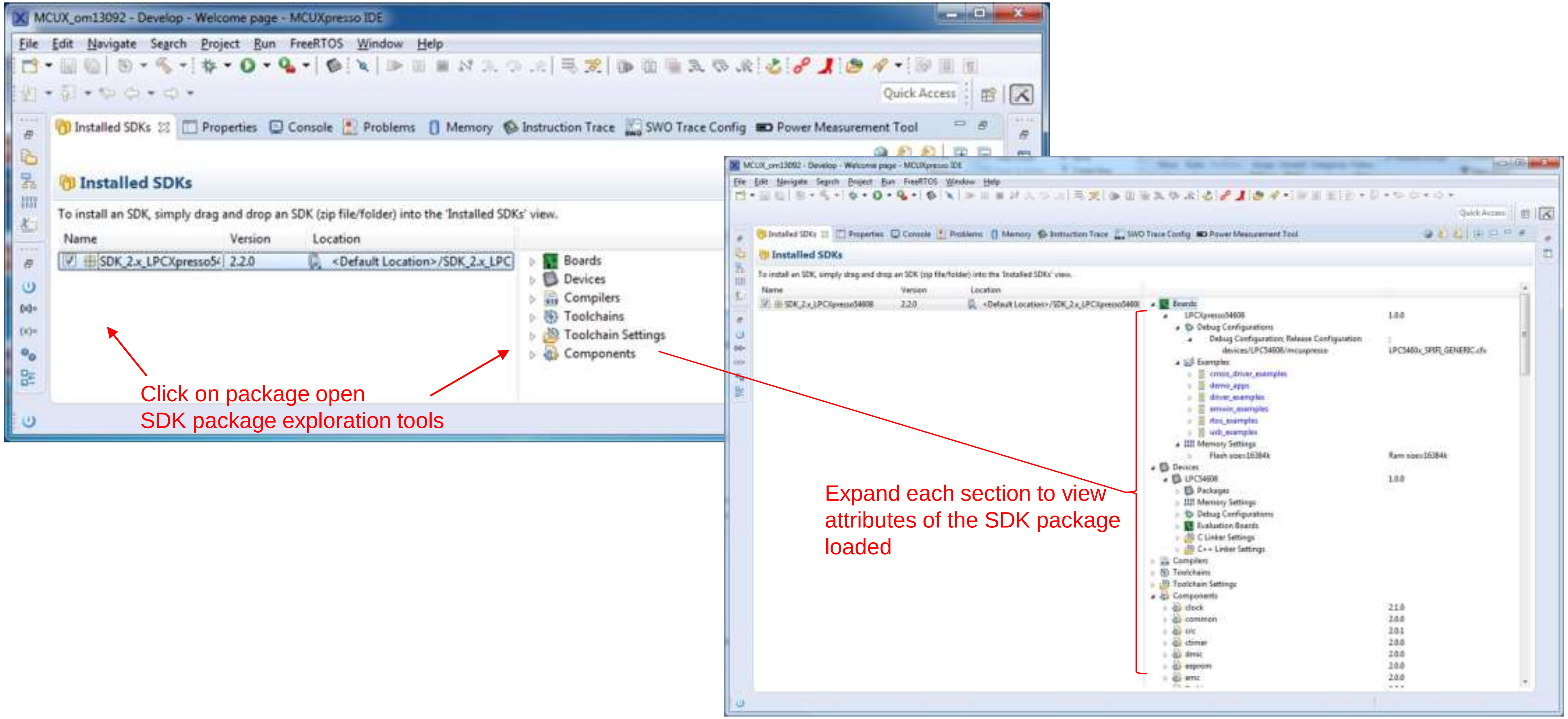
SDK installation process

- 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



MCUX_om13092 - Develop - Welcome page - MCUXpresso IDE
File Edit Navigate Search Project Run FreeRTOS Window Help
Quick Access
Installed SDKs
Properties
Console
Problems
Memory
Instruction Trace
SWO Trace Config
Power Measurement Tool

Installed SDKs

To install an SDK, simply drag and drop an SDK (zip file/folder) into the 'Installed SDKs' view.

Name	Version	Location
☑ SDK_2.x_LPCXpresso54608	2.2.0	<Default Location>/SDK_2.x_LPCXpresso54608

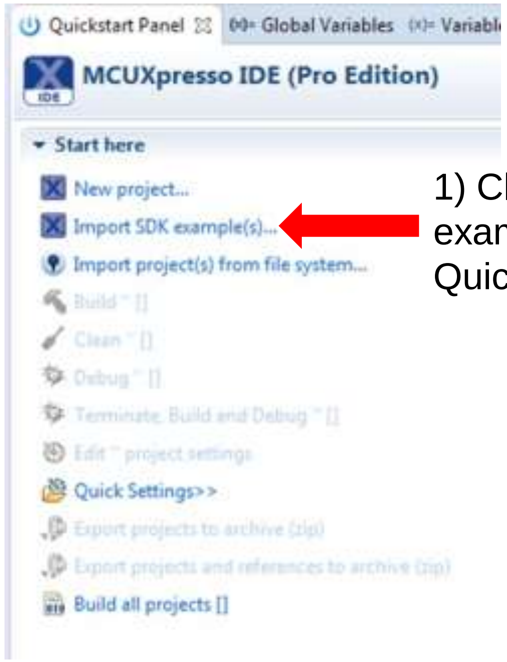
```
▸ Boards
▸ Devices
▸ Compilers
▸ Toolchains
▸ Toolchain Settings
▸ Components
```

```
◢ Boards
   ◢ LPCXpresso54608                                      1.0.0
      ◢ Debug Configurations
         ◢ Debug Configuration; Release Configuration     1
              devices/LPC54608/mcuxpresso                 LPC5460x_SPIFI_GENERIC.cfx
      ◢ Examples
           cmsis_driver_examples
           demo_apps
           driver_examples
           emwin_examples
           rtos_examples
           usb_examples
      ◢ Memory Settings
           Flash size:16384k                              Ram size:16384k
◢ Devices
   ◢ LPC54608                                             1.0.0
      ▸ Packages
      ▸ Memory Settings
      ▸ Debug Configurations
      ▸ Evaluation Boards
      ▸ C Linker Settings
      ▸ C++ Linker Settings
▸ Compilers
▸ Toolchains
▸ Toolchain Settings
◢ Components
   ▸ clock                                                2.1.0
   ▸ common                                               2.0.0
   ▸ crc                                                  2.0.1
   ▸ ctimer                                               2.0.0
   ▸ dmic                                                 2.0.0
   ▸ eeprom                                               2.0.0
   ▸ emc                                                  2.0.0
```

Click on package open
SDK package exploration tools

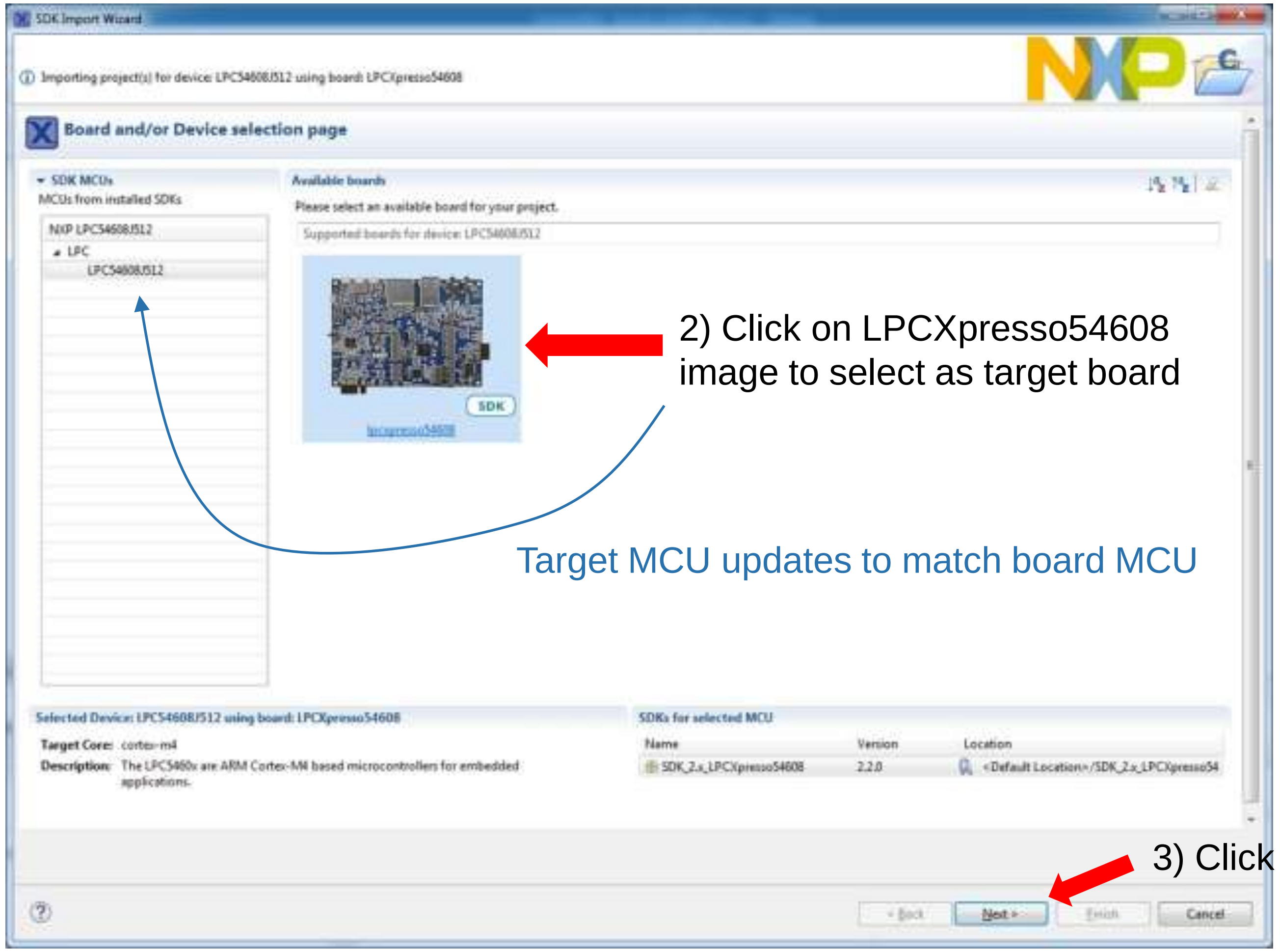
Expand each section to view
attributes of the SDK package
loaded

Follow along: Importing an SDK example



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

SDK Import Dialog Window opens



Importing project(s) for device: LPC54608J512 using board: LPCXpresso54608

MCUs from installed SDKs

NXP LPC54608J512
- LPC
  - LPC54608J512

Please select an available board for your project.

Supported boards for device: LPC54608J512

Selected Device: LPC54608J512 using board: LPCXpresso54608

Target Core: cortex-m4

Description: The LPC5460x are ARM Cortex-M4 based microcontrollers for embedded applications.

SDKs for selected MCU

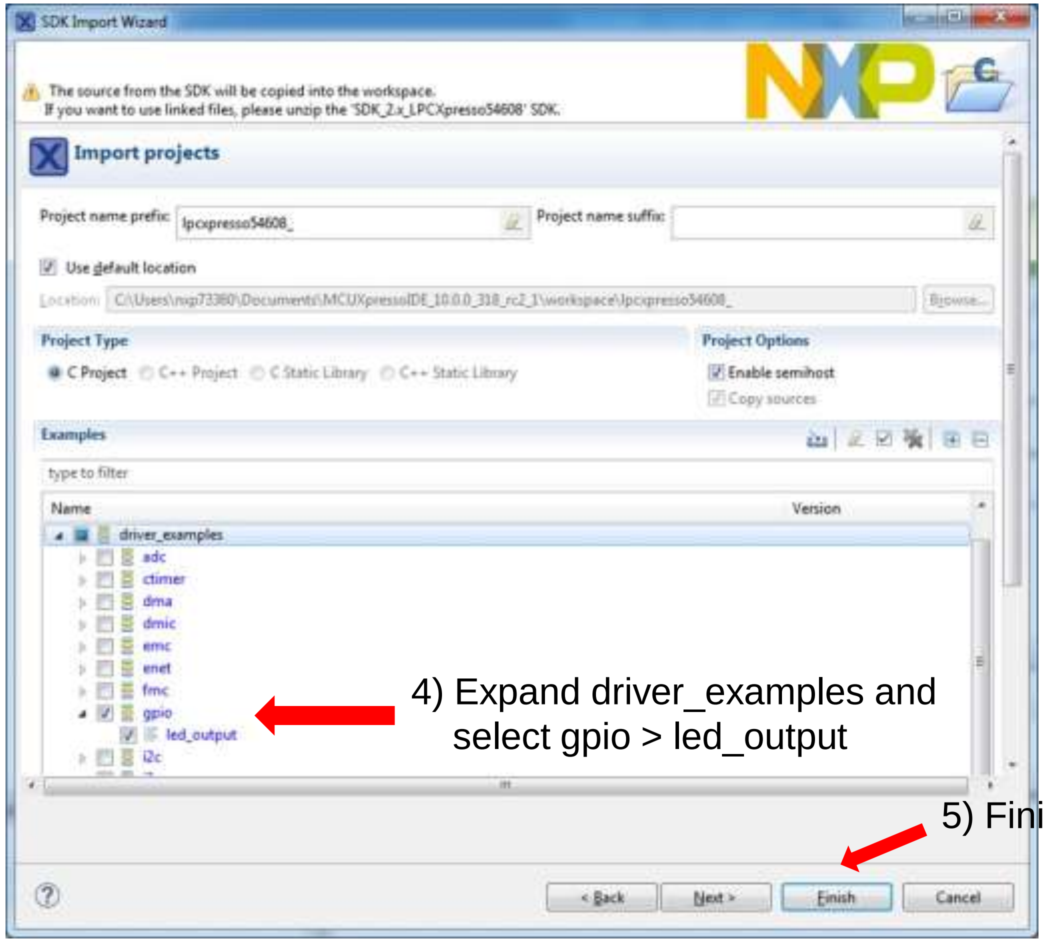
Name	Version	Location
SDK_2.x_LPCXpresso54608	2.2.0	<Default Location>/SDK_2.x_LPCXpresso54

2) Click on LPCXpresso54608 image to select as target board

Target MCU updates to match board MCU

3) Click next

Follow along: Importing an SDK example



The source from the SDK will be copied into the workspace.
If you want to use linked files, please unzip the 'SDK_2.x_LPCXpresso54608' SDK.

Import projects

Project name prefix: lpcxpresso54608_
Project name suffix:

[✓] Use default location

Location: C:\Users\nxp73380\Documents\MCUXpressoIDE_10.0.0_318_rc2_1\workspace\lpcxpresso54608_

Project Type: [✓ C Project] C++ Project C Static Library C++ Static Library

Project Options: [✓] Enable semihost, [✓] Copy sources

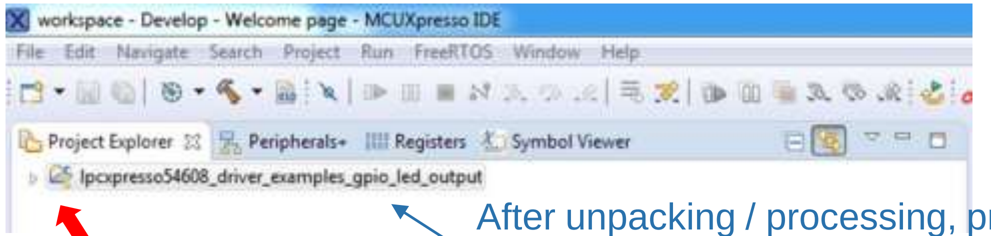
Examples

type to filter

Name	Version
driver_examples	
adc	
ctimer	
dma	
dmic	
emc	
enet	
fmc	
[✓] gpio	
[✓] led_output	
i2c	

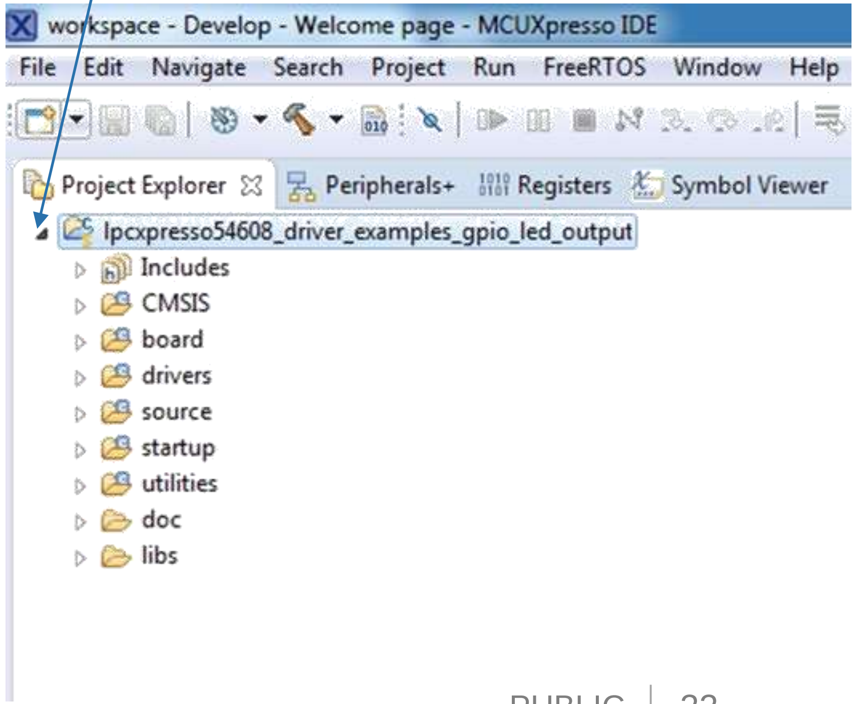
4) Expand driver_examples and select gpio > led_output

5) Finish

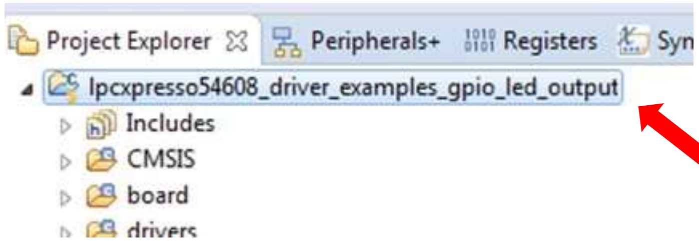

workspace - Develop - Welcome page - MCUXpresso IDE
File Edit Navigate Search Project Run FreeRTOS Window Help
Project Explorer
Peripherals+
Registers
Symbol Viewer
lpcxpresso54608_driver_examples_gpio_led_output

After unpacking / processing, project appears in the project explorer window

6) Click to expand project content

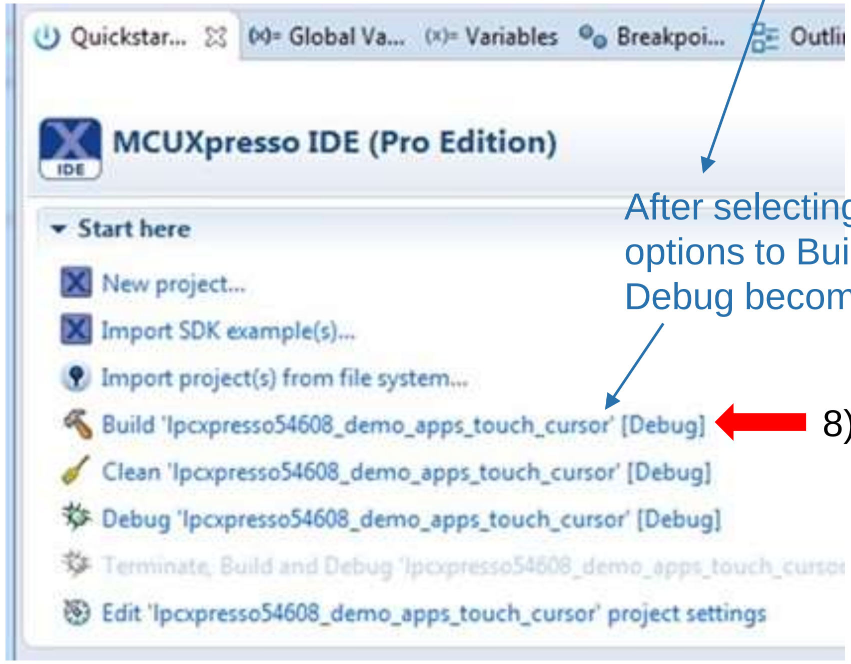

workspace - Develop - Welcome page - MCUXpresso IDE
File Edit Navigate Search Project Run FreeRTOS Window Help
Project Explorer
Peripherals+
Registers
Symbol Viewer
lpcxpresso54608_driver_examples_gpio_led_output
Includes
CMSIS
board
drivers
source
startup
utilities
doc
libs

Follow along: Building SDK example

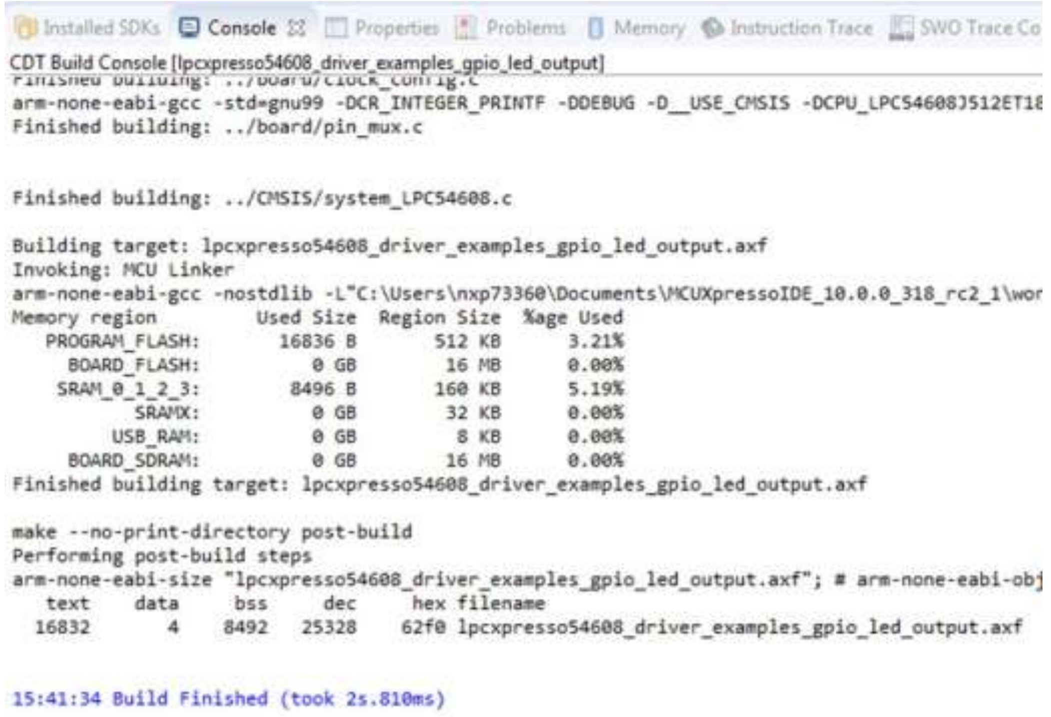


7) Click on the project top level to make it active

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


Quickstar...
Global Va...
Variables
Breakpoi...
Outli
MCUXpresso IDE (Pro Edition)
Start here
New project...
Import SDK example(s)...
Import project(s) from file system...
Build 'lpcxpresso54608_demo_apps_touch_cursor' [Debug]
Clean 'lpcxpresso54608_demo_apps_touch_cursor' [Debug]
Debug 'lpcxpresso54608_demo_apps_touch_cursor' [Debug]
Terminate, Build and Debug 'lpcxpresso54608_demo_apps_touch_curso
Edit 'lpcxpresso54608_demo_apps_touch_cursor' project settings

8) Click Build



```
CDT Build Console [lpcxpresso54608_driver_examples_gpio_led_output]
Finished building: ../board/clock_config.c
arm-none-eabi-gcc -std=gnu99 -DCR_INTEGER_PRINTF -DDEBUG -D__USE_CMSIS -DCPU_LPC54608J512ET18
Finished building: ../board/pin_mux.c


Finished building: ../CMSIS/system_LPC54608.c

Building target: lpcxpresso54608_driver_examples_gpio_led_output.axf
Invoking: MCU Linker
arm-none-eabi-gcc -nostdlib -L"C:\Users\nxp73360\Documents\MCUXpressoIDE_10.0.0_318_rc2_1\wor
Memory region         Used Size  Region Size  %age Used
   PROGRAM_FLASH:       16836 B       512 KB      3.21%
     BOARD_FLASH:          0 GB        16 MB      0.00%
     SRAM_0_1_2_3:       8496 B       160 KB      5.19%
           SRAMX:          0 GB        32 KB      0.00%
         USB_RAM:          0 GB         8 KB      0.00%
     BOARD_SDRAM:          0 GB        16 MB      0.00%
Finished building target: lpcxpresso54608_driver_examples_gpio_led_output.axf

make --no-print-directory post-build
Performing post-build steps
arm-none-eabi-size "lpcxpresso54608_driver_examples_gpio_led_output.axf"; # arm-none-eabi-obj
   text    data     bss     dec     hex filename
  16832       4    8492   25328    62f0 lpcxpresso54608_driver_examples_gpio_led_output.axf


15:41:34 Build Finished (took 2s.810ms)
```

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

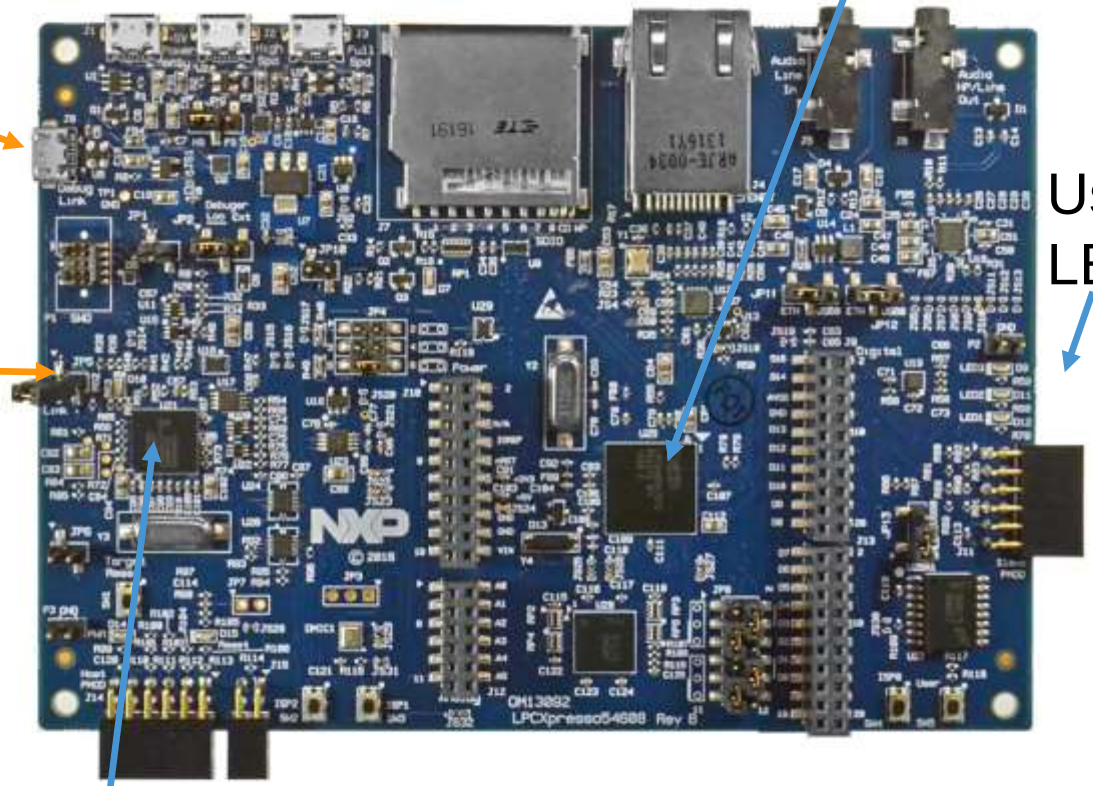
LPCXpresso54608 Board Configuration

LPC54608 target MCU (Cortex-M4F)

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

User LEDs

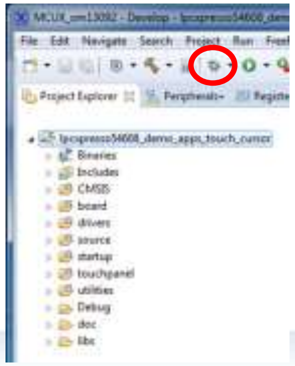
Before connecting to USB, ensure that “DFU Link” (JP5) jumper is fitted so that MCUXpresso IDE can soft-load probe firmware

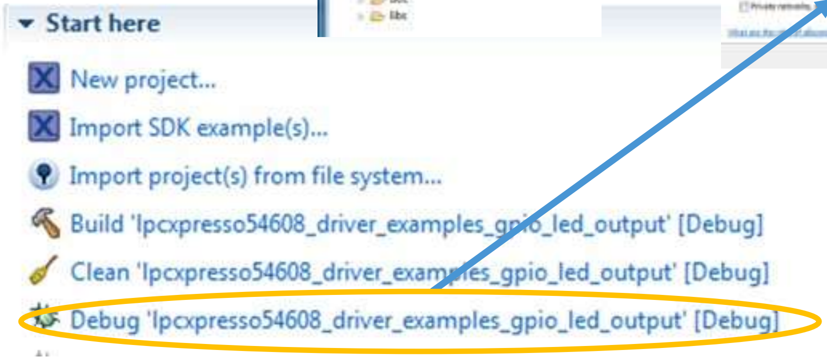


LPC-Link2 debug probe MCU

Follow along: Start Debug Session

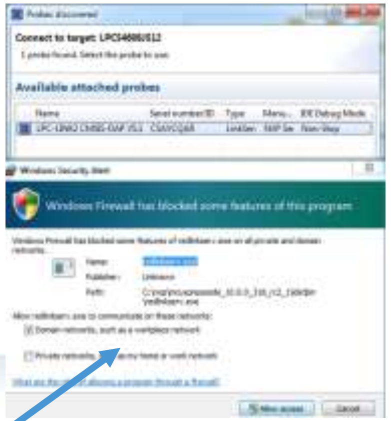
Note: use Quickstart Debug, not this icon





Start here
New project...
Import SDK example(s)...
Import project(s) from file system...
Build 'lpcxpresso54608_driver_examples_gpio_led_output' [Debug]
Clean 'lpcxpresso54608_driver_examples_gpio_led_output' [Debug]
Debug 'lpcxpresso54608_driver_examples_gpio_led_output' [Debug]

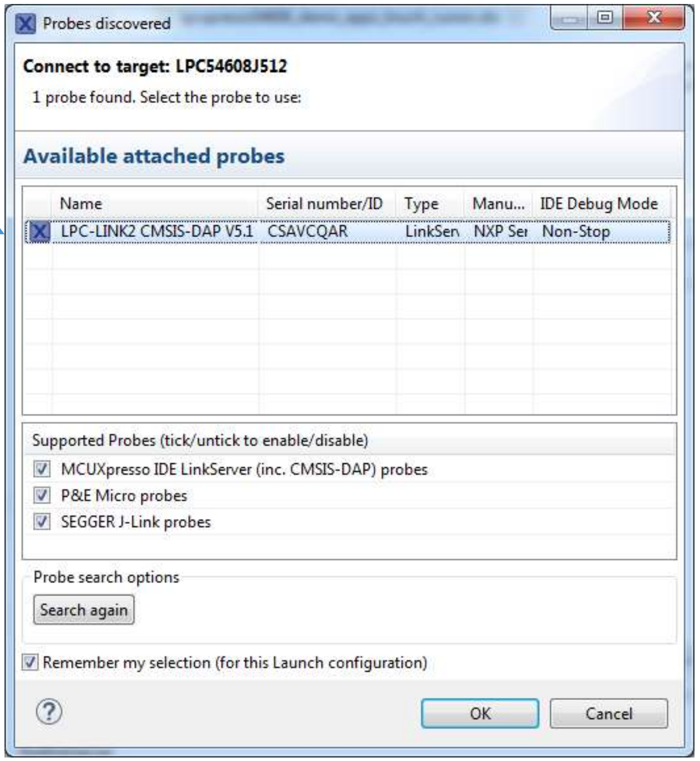
Select project in the Project Explorer View, then click on Debug in the Quickstart Panel



Probes discovered
Connect to target: LPC54608J512
Available attached probes
LPC-LINK2 CMSIS-DAP V5.1 CSAVCQAR
Windows Security Alert
Windows Firewall has blocked some features of this program
Domain networks, such as a workplace network
Private networks, such as my home or work network
Allow access
Cancel

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

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


Probes discovered

Connect to target: LPC54608J512

1 probe found. Select the probe to use:

Available attached probes

Name	Serial number/ID	Type	Manu...	IDE Debug Mode
LPC-LINK2 CMSIS-DAP V5.1	CSAVCQAR	LinkSen	NXP Ser	Non-Stop

Supported Probes (tick/untick to enable/disable)

- [x] MCUXpresso IDE LinkServer (inc. CMSIS-DAP) probes
- [x] P&E Micro probes
- [x] SEGGER J-Link probes

Probe search options

Search again

- [x] Remember my selection (for this Launch configuration)

OK Cancel

These settings will be remembered for next time you debug this project

Click OK

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

Follow along: Develop Perspective – Debugging

Run Controls

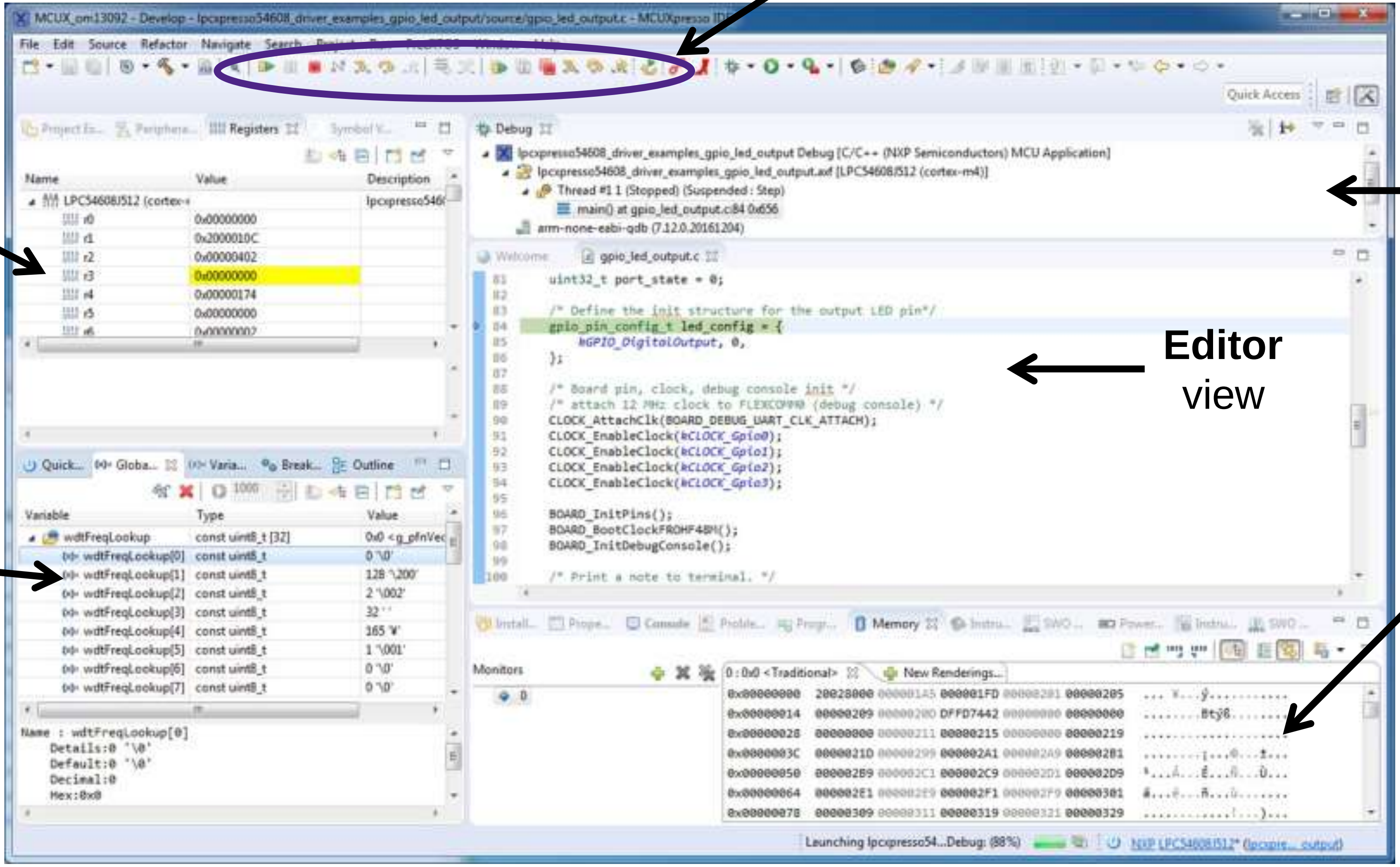
Registers and **Peripherals** views

Debug view

Editor view

Variables, Global Variables and **Breakpoints** views

Console, Memory and **Trace** views


MCUX_om13092 - Develop - lpcxpresso54608_driver_examples_gpio_led_output/source/gpio_led_output.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search
Quick Access
Project Ex... Periphera... Registers Symbol V...
Quick... Globa... Varia... Break... Outline
Debug
Welcome gpio_led_output.c
Install... Prope... Console Proble... Progr... Memory Instru... SWO ... Power... Instru... SWO ...
Monitors
0 : 0x0 <Traditional> New Renderings...

Name	Value	Description
LPC54608J512 (cortex-m4)		lpcxpresso546
r0	0x00000000	
r1	0x2000010C	
r2	0x00000402	
r3	0x00000000	
r4	0x00000174	
r5	0x00000000	
r6	0x00000002	

```
lpcxpresso54608_driver_examples_gpio_led_output Debug [C/C++ (NXP Semiconductors) MCU Application]
  lpcxpresso54608_driver_examples_gpio_led_output.axf [LPC54608J512 (cortex-m4)]
    Thread #1 1 (Stopped) (Suspended : Step)
      main() at gpio_led_output.c:84 0x656
  arm-none-eabi-gdb (7.12.0.20161204)
```

```c
    uint32_t port_state = 0;

    /* Define the init structure for the output LED pin*/
    gpio_pin_config_t led_config = {
        kGPIO_DigitalOutput, 0,
    };

    /* Board pin, clock, debug console init */
    /* attach 12 MHz clock to FLEXCOMM0 (debug console) */
    CLOCK_AttachClk(BOARD_DEBUG_UART_CLK_ATTACH);
    CLOCK_EnableClock(kCLOCK_Gpio0);
    CLOCK_EnableClock(kCLOCK_Gpio1);
    CLOCK_EnableClock(kCLOCK_Gpio2);
    CLOCK_EnableClock(kCLOCK_Gpio3);

    BOARD_InitPins();
    BOARD_BootClockFROHF48M();
    BOARD_InitDebugConsole();

    /* Print a note to terminal. */
```

Variable	Type	Value
wdtFreqLookup	const uint8_t [32]	0x0 <g_pfnVec
wdtFreqLookup[0]	const uint8_t	0 '\0'
wdtFreqLookup[1]	const uint8_t	128 '\200'
wdtFreqLookup[2]	const uint8_t	2 '\002'
wdtFreqLookup[3]	const uint8_t	32 ' '
wdtFreqLookup[4]	const uint8_t	165 '¥'
wdtFreqLookup[5]	const uint8_t	1 '\001'
wdtFreqLookup[6]	const uint8_t	0 '\0'
wdtFreqLookup[7]	const uint8_t	0 '\0'

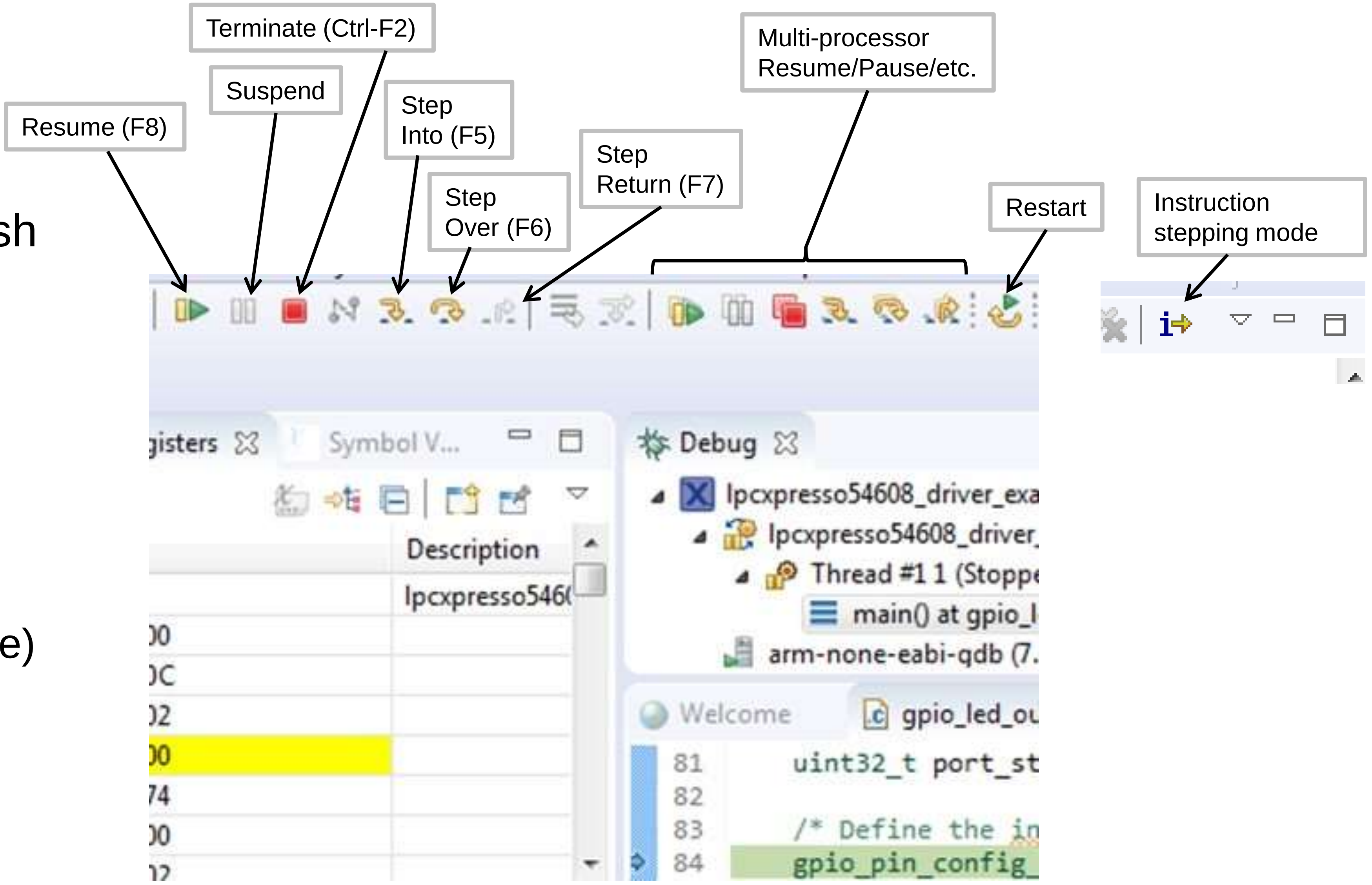
```
Name : wdtFreqLookup[0]
    Details:0 '\0'
    Default:0 '\0'
    Decimal:0
    Hex:0x0
```

```
0x00000000  20028000 000001A5 000001FD 00000201 00000205
0x00000014  00000209 0000020D DFFD7442 00000000 00000000
0x00000028  00000000 00000211 00000215 00000000 00000219
0x0000003C  0000021D 00000299 000002A1 000002A9 000002B1
0x00000050  000002B9 000002C1 000002C9 000002D1 000002D9
0x00000064  000002E1 000002E9 000002F1 000002F9 00000301
0x00000078  00000309 00000311 00000319 00000321 00000329
```

Launching lpcxpresso54...Debug: (88%) NXP LPC54608J512* (lpcxpre... output)

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



```
uint32_t port_st

/* Define the in
gpio_pin_config_
```

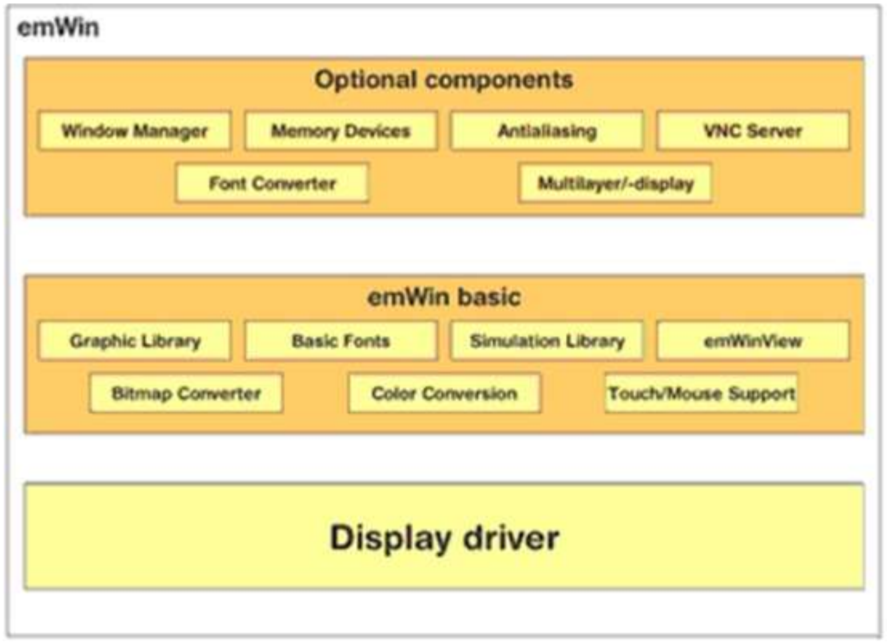
Follow along: Develop Perspective – Debugging

- 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

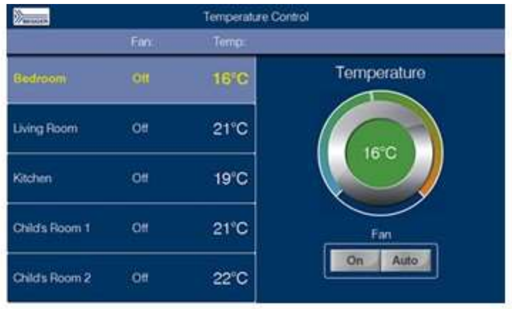
emWin Introduction

- 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


Temperature Control

	Fan:	Temp:
Bedroom	Off	16°C
Living Room	Off	21°C
Kitchen	Off	19°C
Child's Room 1	Off	21°C
Child's Room 2	Off	22°C

- 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)

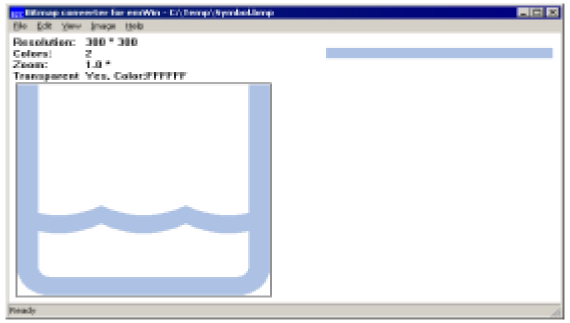
emWIN is included in MCUXpresso SDK packages (for LPC546xx)

What NXP customers used emWin for…

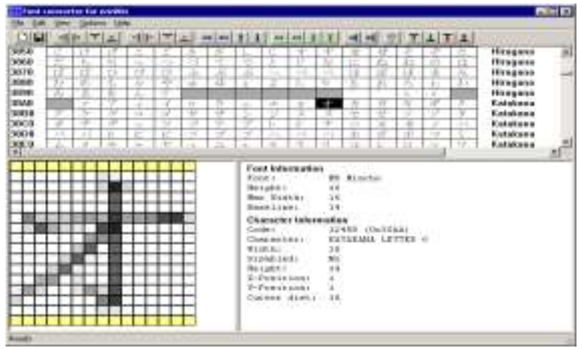
- Paper money counter (LPC4300 + 3.1” LCD)
- ATM (LPC1800 + 14”, 1024x768 LCD)
- Industrial touch panel (LPC1788 + 10.1”, 640x480 LCD)
- Washing machine (LPC3000)
- Elevator control with LCD (LPC1788)
- High accuracy scales with LCD (LPC1788)
- Security Panel (LPC2132)

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 a 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.


Bitmap converter for emWin - C:\Temp\Symbol.bmp
File Edit View Image Help
Ready

Resolution: 300 * 300
Colors: 2
Zoom: 1.0 *
Transparent Yes, Color:FFFFFF



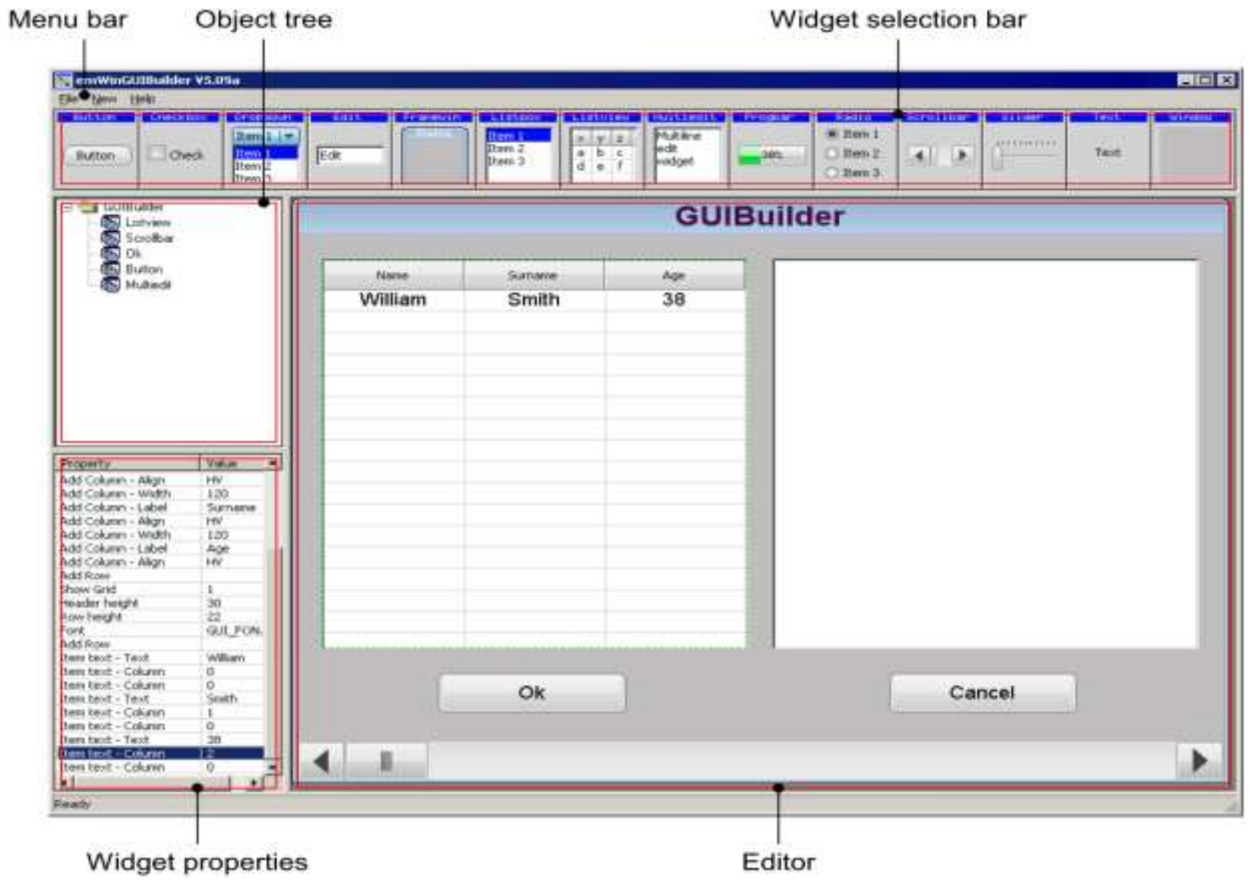
GUI-Builder

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

Basic usage of the GUI-Builder:

- **Check project path** in `GUIBuilder.ini`
  This file can be found in the application folder of the tool.
- **Start GUI-Builder**
- **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`



Name	Surname	Age
William	Smith	38

Property	Value
Add Column - Align	HV
Add Column - Width	120
Add Column - Label	Surname
Add Column - Align	HV
Add Column - Width	120
Add Column - Label	Age
Add Column - Align	HV
Add Row	
Show Grid	1
Header height	30
Row height	22
Font	GUI_FON
Add Row	
Item text - Text	William
Item text - Column	0
Item text - Column	0
Item text - Text	Smith
Item text - Column	1
Item text - Column	0
Item text - Text	38
Item text - Column	2
Item text - Column	0

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()`

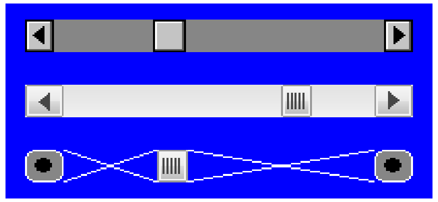
- The skin of each single wigdet can be set by:

`<WIDGET>_SetSkin()`

- Properties of FLEX_SKIN can be fetched / set by:

`<WIDGET>_GetSkinFlexProps()`

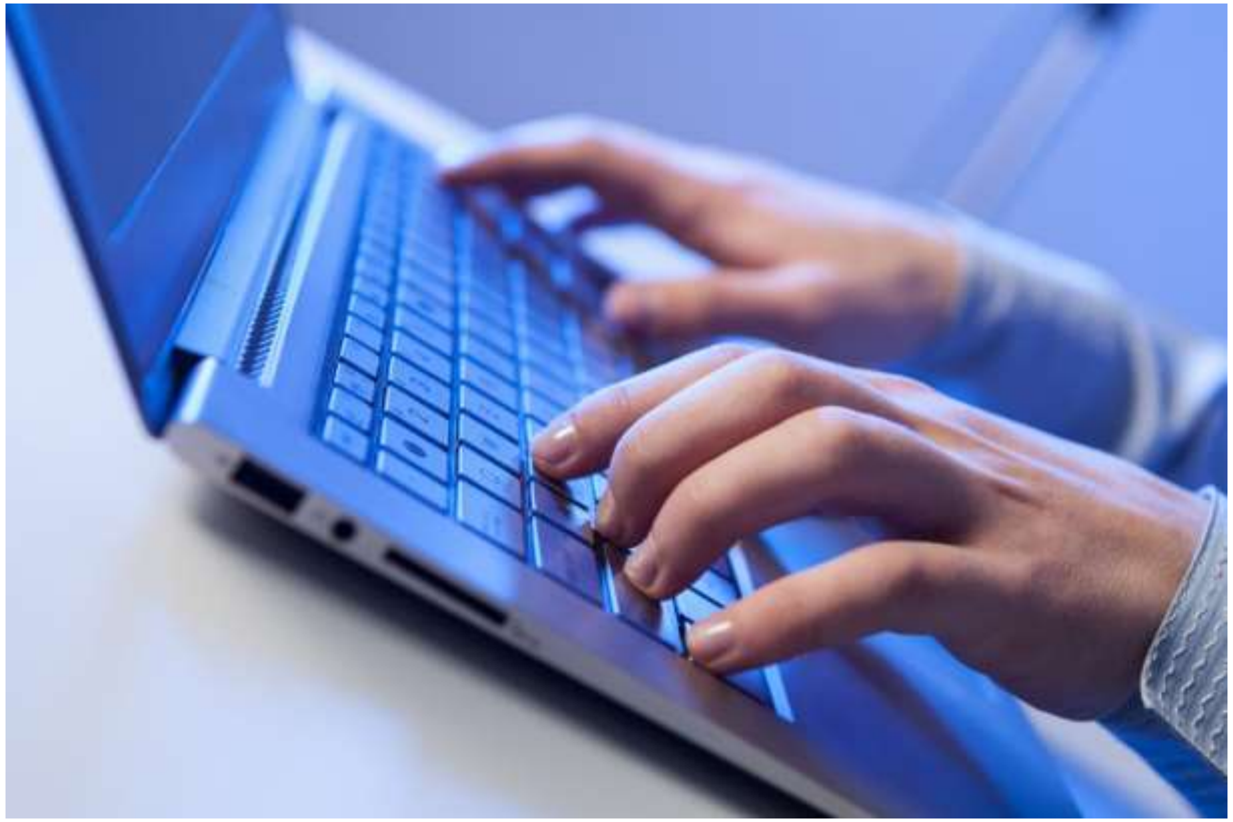
`<WIDGET>_SetSkinFlexProps()`



```
#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

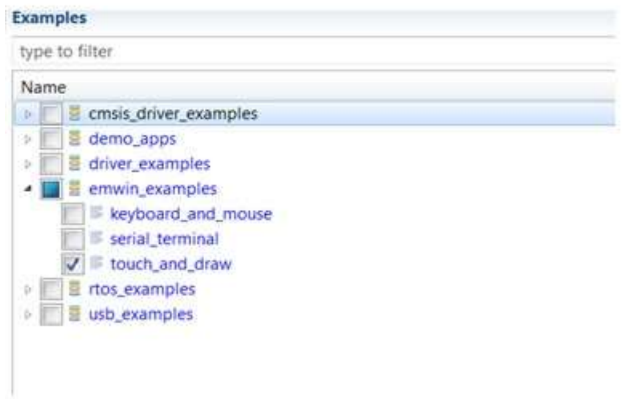
Hands-on – Session #2



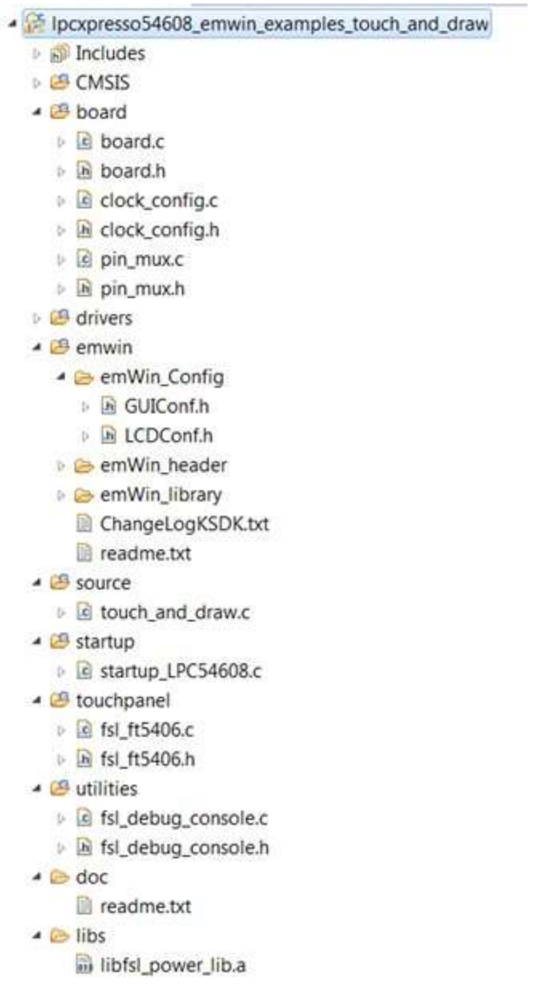
- Hands-on session with following objectives:
  - Build and run an emWin example from the LPC54608 MCUXpresso SDK
  - Modify the example to add a logo
  - Create a simple GUI using GUIBuilder tools

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.



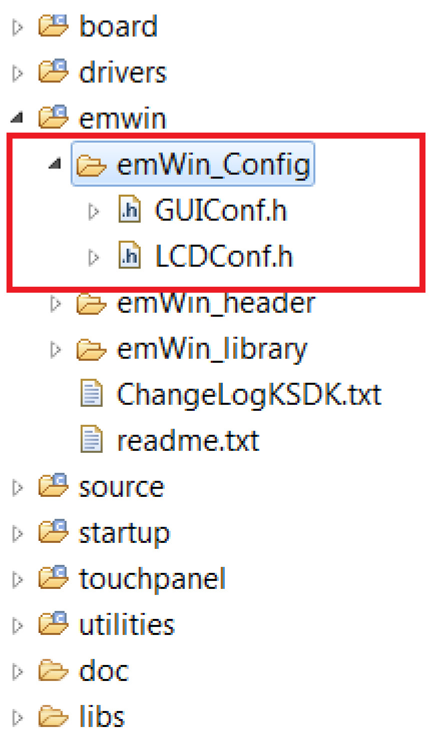
```
Examples
type to filter
Name
▷ ☐ cmsis_driver_examples
▷ ☐ demo_apps
▷ ☐ driver_examples
◢ ■ emwin_examples
      ☐ keyboard_and_mouse
      ☐ serial_terminal
      ✓ touch_and_draw
▷ ☐ rtos_examples
▷ ☐ usb_examples
```



```
◢ lpcxpresso54608_emwin_examples_touch_and_draw
   ▷ Includes
   ▷ CMSIS
   ◢ board
      ▷ board.c
      ▷ board.h
      ▷ clock_config.c
      ▷ clock_config.h
      ▷ pin_mux.c
      ▷ pin_mux.h
   ▷ drivers
   ◢ emwin
      ◢ emWin_Config
         ▷ GUIConf.h
         ▷ LCDConf.h
      ▷ emWin_header
      ▷ emWin_library
        ChangeLogKSDK.txt
        readme.txt
   ◢ source
      ▷ touch_and_draw.c
   ◢ startup
      ▷ startup_LPC54608.c
   ◢ touchpanel
      ▷ fsl_ft5406.c
      ▷ fsl_ft5406.h
   ◢ utilities
      ▷ fsl_debug_console.c
      ▷ fsl_debug_console.h
   ◢ doc
        readme.txt
   ◢ libs
        libfsl_power_lib.a
```

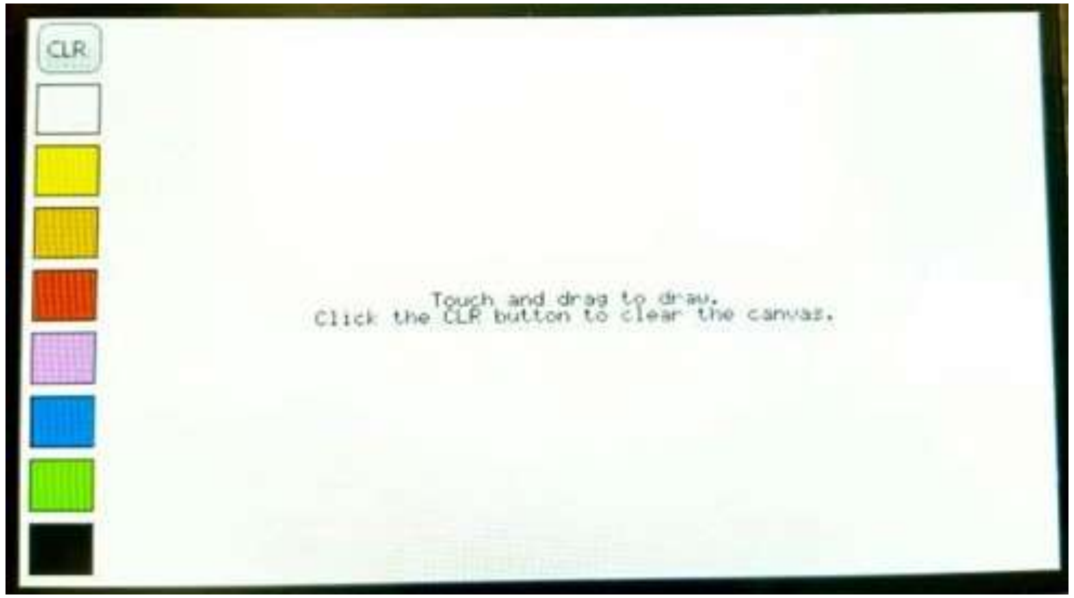
emWin Configuration

- 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.

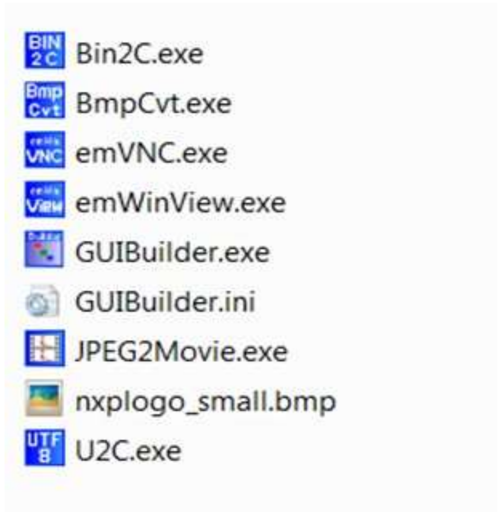


Change Background Bitmap Image

- 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

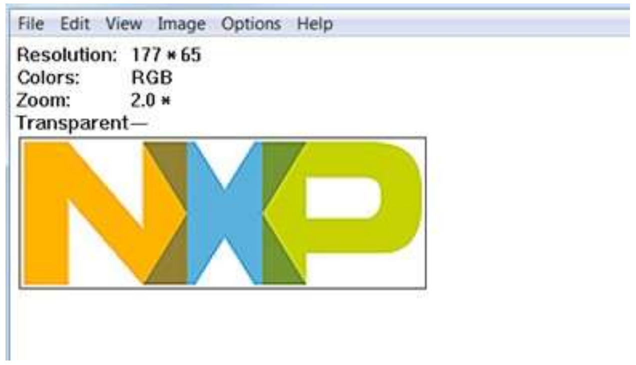
- To create a new bitmap to use as the display background, you have two choices:
  - Use the available NXP logo provided in the project folder. All instructions are written for this case, so this is the easiest path.
- Navigate the folder emWin_Tools in the package. This folder contains various emWin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



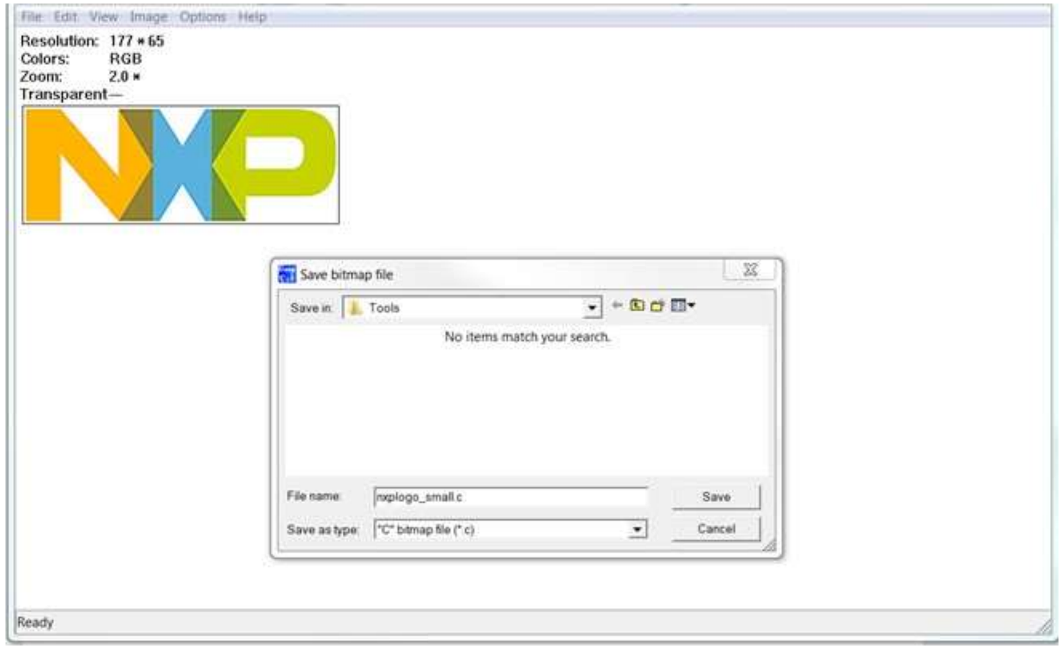
Bin2C.exe
BmpCvt.exe
emVNC.exe
emWinView.exe
GUIBuilder.exe
GUIBuilder.ini
JPEG2Movie.exe
nxplogo_small.bmp
U2C.exe

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.



Resolution: 177 × 65
Colors: RGB
Zoom: 2.0 ×
Transparent—

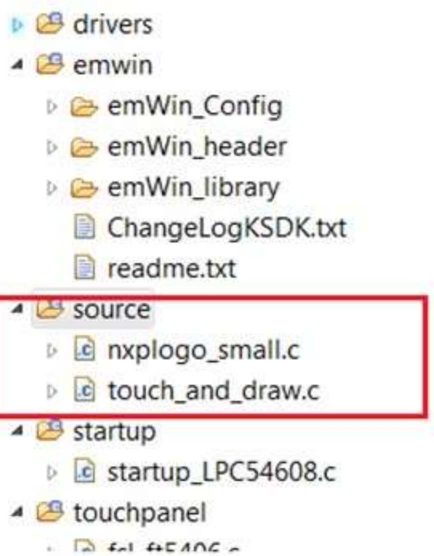


Resolution: 177 × 65
Colors: RGB
Zoom: 2.0 ×
Transparent—

Save in: Tools
No items match your search.
File name: nxplogo_small.c
Save as type: "C" bitmap file (*.c)

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.

    ```
    extern const GUI_BITMAP bmnxplogo_small;
    ```

  - Paste the following as first statement inside for(;;) loop.

    ```
    GUI_DrawBitmap(&bmnxplogo_small, 130, 0);

    Example:
    for (;;)
    {
        GUI_DrawBitmap(&bmnxplogo_small, 130, 0);
        ...
        ...
    }
    ```

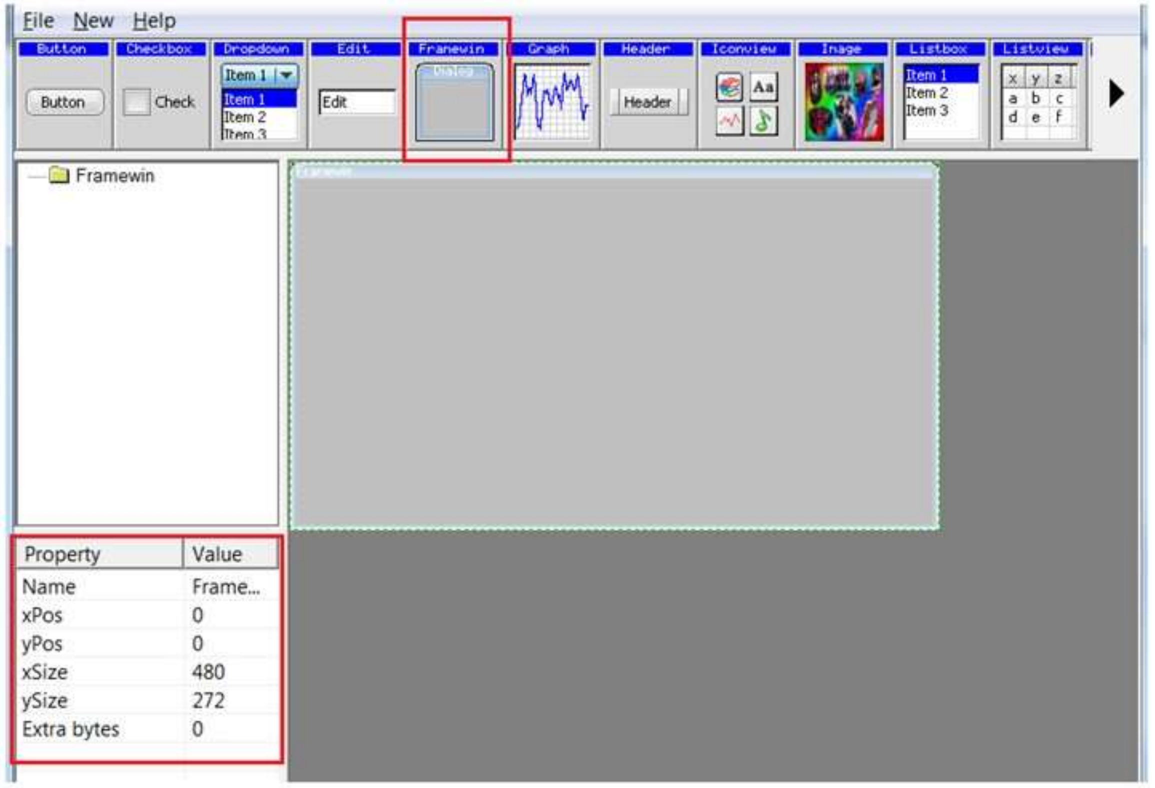
- Build and Debug the application. Observe the updated bitmap displayed in the background of the frame.

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.



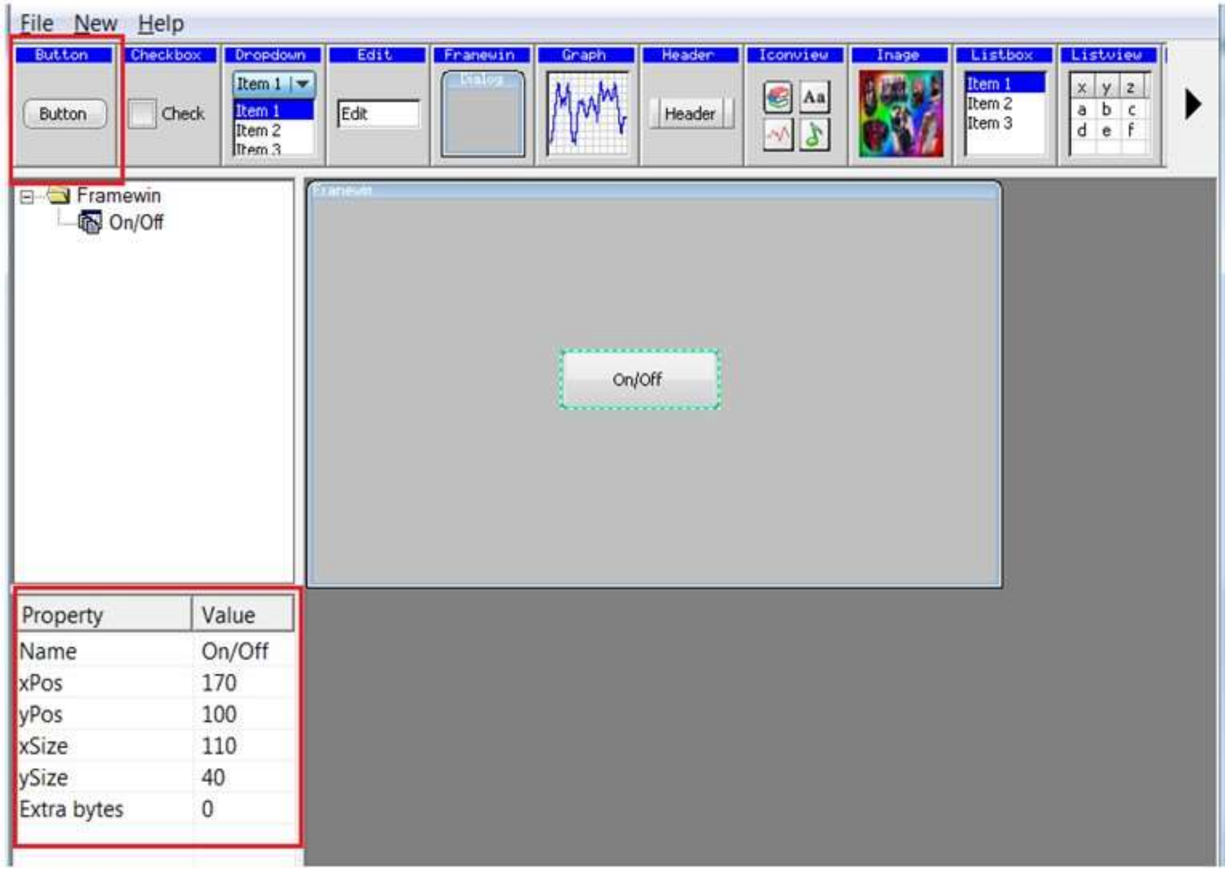
Property	Value
Name	Frame...
xPos	0
yPos	0
xSize	480
ySize	272
Extra bytes	0

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.

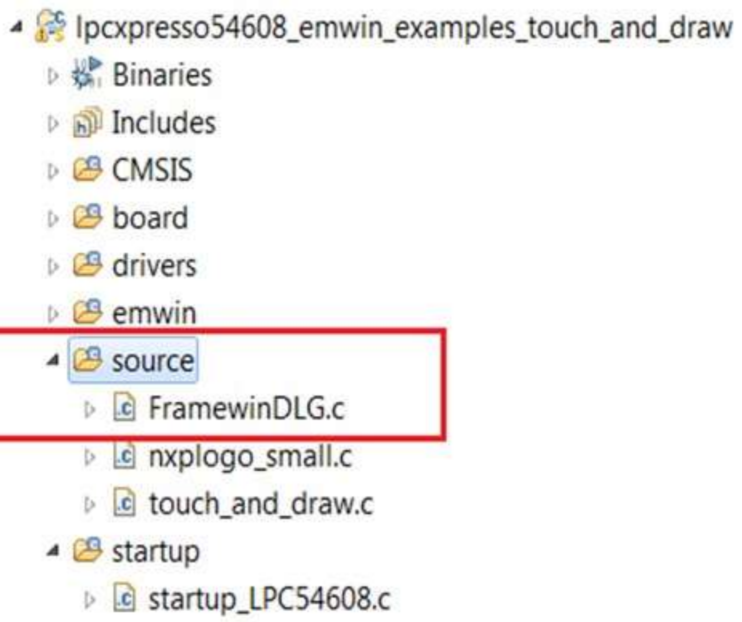
- Save the GUI by clicking **File -> Save**. This creates the file **FramewinDLG.c** in the emWin_Tools folder (refer to path specified in the GUIBuilder.ini file).



Property	Value
Name	On/Off
xPos	170
yPos	100
xSize	110
ySize	40
Extra bytes	0

Follow Along: Add FramewinDLG.c file to project workspace

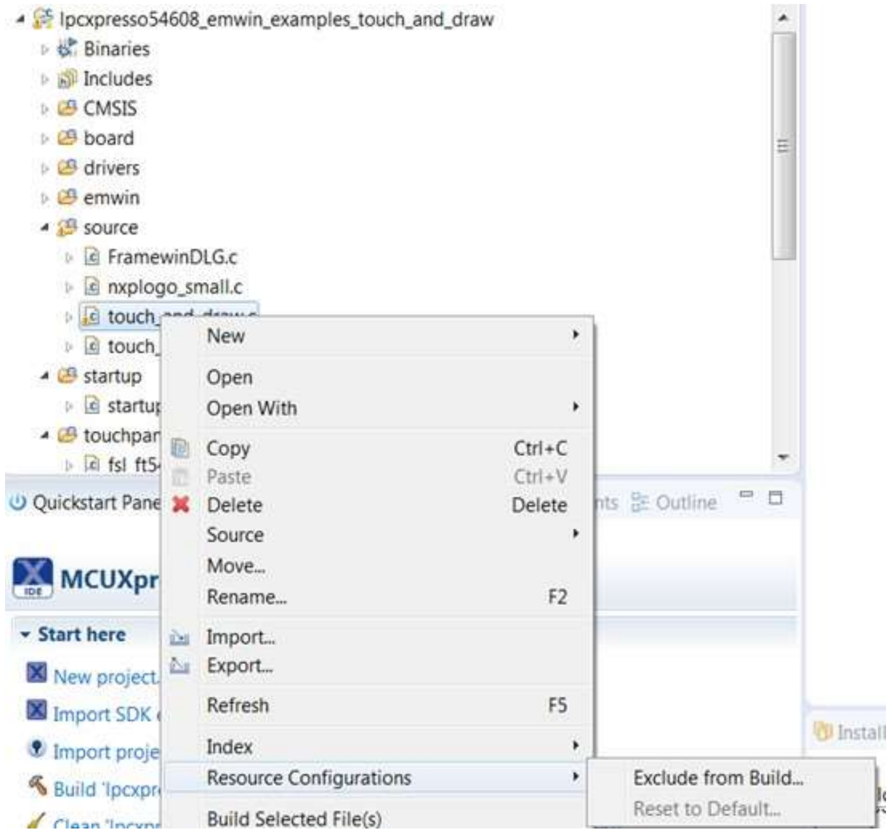
- To do this either drag and drop the FramewinDLG.c file into the source folder of the project in workspace or copy the FramewinDLG.c and paste it in the **source** folder of the workspace.



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().

```
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.

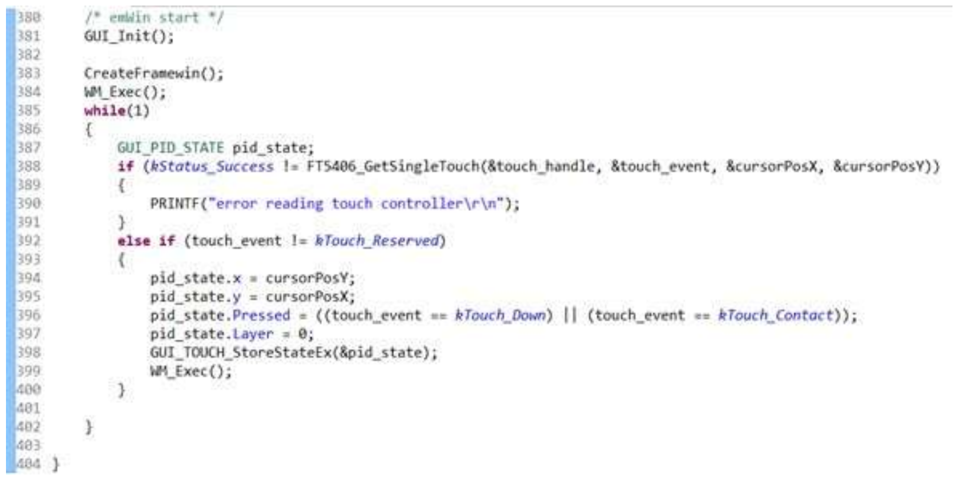
Follow Along : Add code to Control an LED

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

```
LED3_INIT(0);
```

- Copy paste the following lines after GUI_Init().

```
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();
      }

    }
```



```
    /* emWin start */
    GUI_Init();

    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:

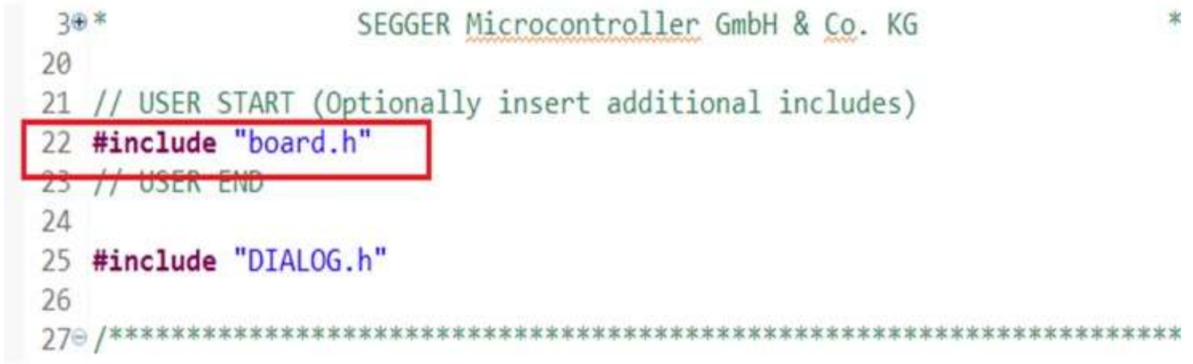
  `#include “board.h”`

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

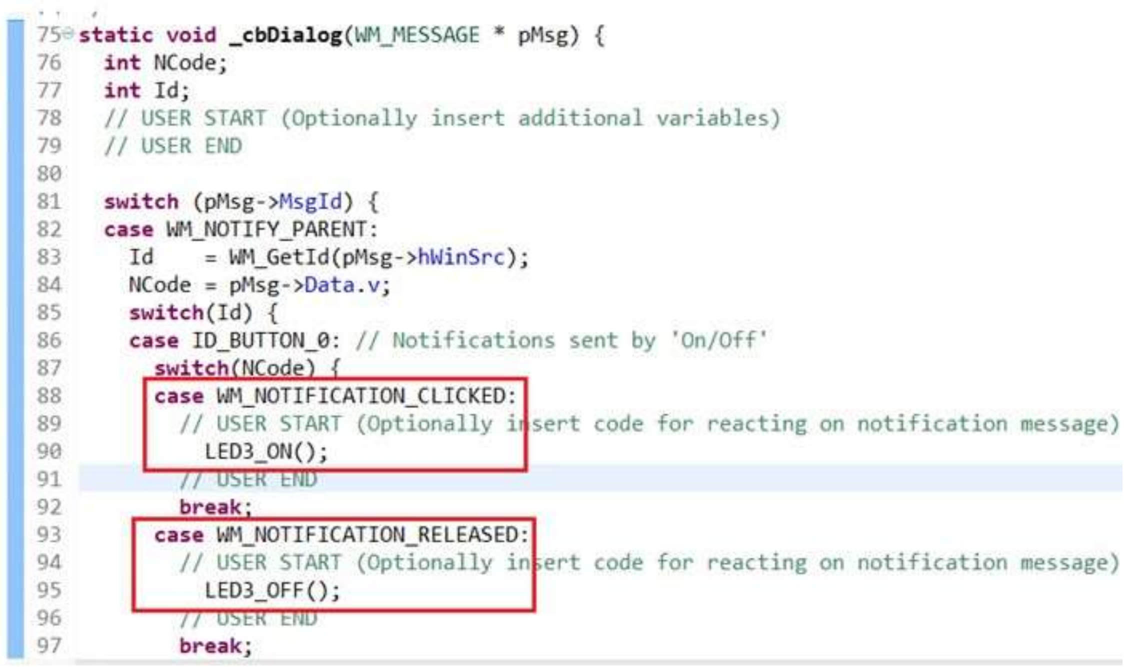
  CLICKED : `LED3_ON();`

  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.



```
3⊕*                    SEGGER Microcontroller GmbH & Co. KG                    *
20
21 // USER START (Optionally insert additional includes)
22 #include "board.h"
23 // USER END
24
25 #include "DIALOG.h"
26
27⊖/*********************************************************************
```

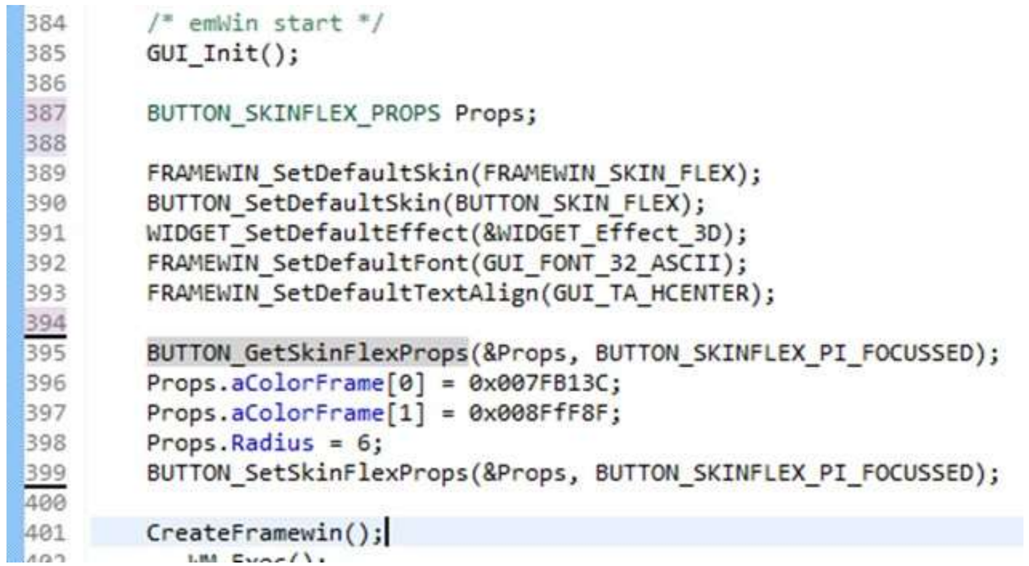


```
75⊖static void _cbDialog(WM_MESSAGE * pMsg) {
76   int NCode;
77   int Id;
78   // USER START (Optionally insert additional variables)
79   // USER END
80
81   switch (pMsg->MsgId) {
82   case WM_NOTIFY_PARENT:
83     Id    = WM_GetId(pMsg->hWinSrc);
84     NCode = pMsg->Data.v;
85     switch(Id) {
86     case ID_BUTTON_0: // Notifications sent by 'On/Off'
87       switch(NCode) {
88       case WM_NOTIFICATION_CLICKED:
89         // USER START (Optionally insert code for reacting on notification message)
90           LED3_ON();
91         // USER END
92         break;
93       case WM_NOTIFICATION_RELEASED:
94         // USER START (Optionally insert code for reacting on notification message)
95           LED3_OFF();
96         // USER END
97         break;
```

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();`
-

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



```
    /* emWin start */
    GUI_Init();

    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_FOCUSSED);
    Props.aColorFrame[0] = 0x007FB13C;
    Props.aColorFrame[1] = 0x008FfF8F;
    Props.Radius = 6;
    BUTTON_SetSkinFlexProps(&Props, BUTTON_SKINFLEX_PI_FOCUSSED);

    CreateFramewin();
```

- Add this include to the top of touchbutton.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 = **Win**dow + Gad**get**

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:

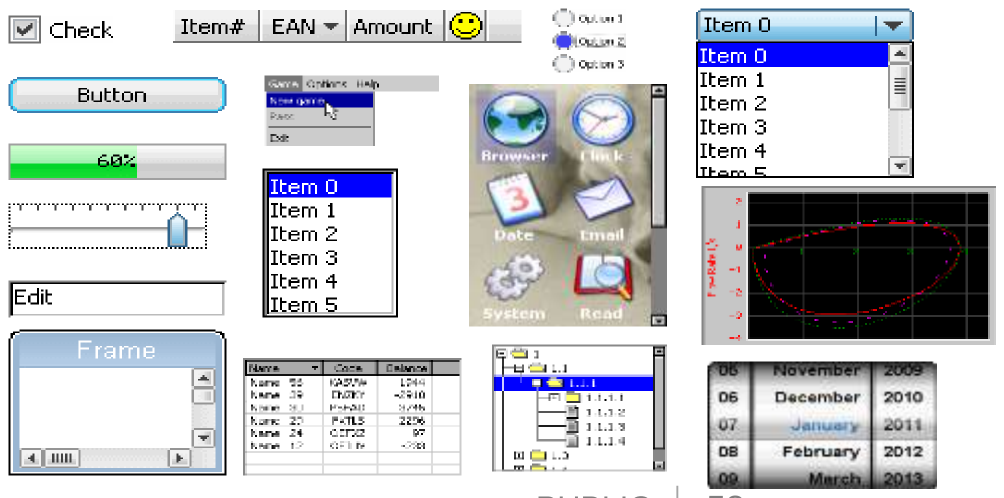
- **Direct creation**
  For each widget there exist creation functions:
  - `<WIDGET>_CreateEx()` : Creation without user data.
  - `<WIDGET>_CreateUser()` : Creation with user data.
- **Indirect creation**
  Indirect means using a dialog box creation function and a `GUI_WIDGET_CREATE_INFO` structure which contains a pointer to the indirect creation routine:
  - `<WIDGET>_CreateIndirect()` : Creation by dialog box creation function.

Direct creation

```
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

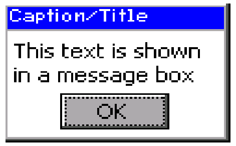
```
#include "DIALOG.h"
static const GUI_WIDGET_CREATE_INFO _aDialogCreate[] = {
  { FRAMEWIN_CreateIndirect, "Window", 0, 10, 10, 100, 50 }
};

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

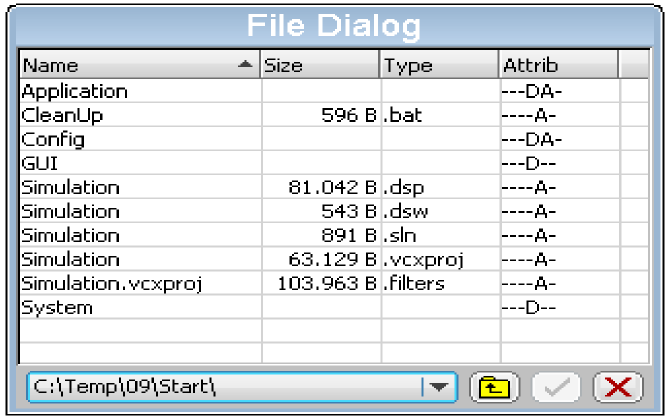
- Common dialogs available:
  - Message boxes
  - Color selection
  - Exploring a file system

- **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.



Caption/Title

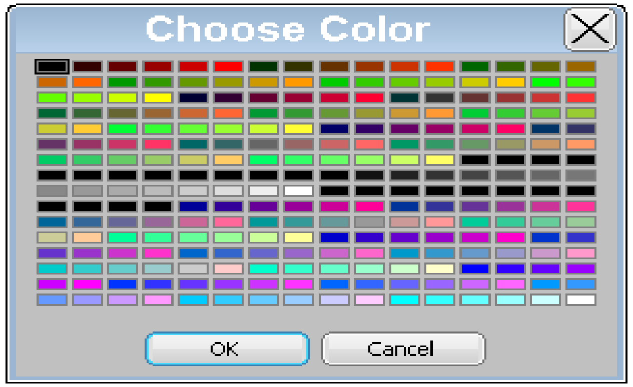
This text is shown in a message box



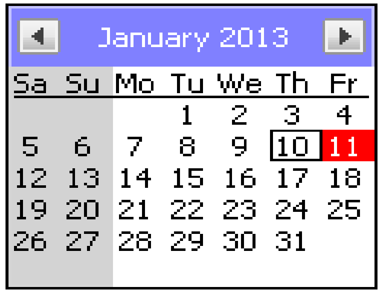
File Dialog

Name	Size	Type	Attrib
Application			---DA-
CleanUp	596 B	.bat	----A-
Config			---DA-
GUI			---D--
Simulation	81.042 B	.dsp	----A-
Simulation	543 B	.dsw	----A-
Simulation	891 B	.sln	----A-
Simulation	63.129 B	.vcxproj	----A-
Simulation.vcxproj	103.963 B	.filters	----A-
System			---D--

C:\Temp\09\Start\



Choose Color



January 2013

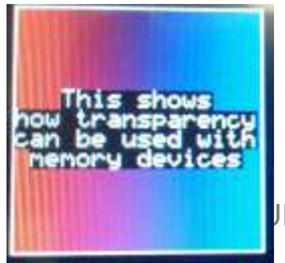
Sa	Su	Mo	Tu	We	Th	Fr
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Memory devices

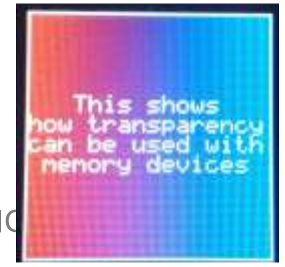
- **How do they work?**
  - Drawing operations can be passed to a memory device instead to the display to prevent display from flickering. A memory device is a hardware independent destination device for drawing operations.
- **What can they be used for?**
  - Preventing flickering
  - Container for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

```
#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\n"
                       "how transparency\n"
                       "can be used with\n"
                       "memory devices"
                       , &Rect
                       , GUI_TA_HCENTER | GUI_TA_VCENTER);
  GUI_MEMDEV_Select(0);
  GUI_MEMDEV_Write(hMem);
  while (1) {
    GUI_Delay(100);
  }
}
```

Without transparency



With transparency



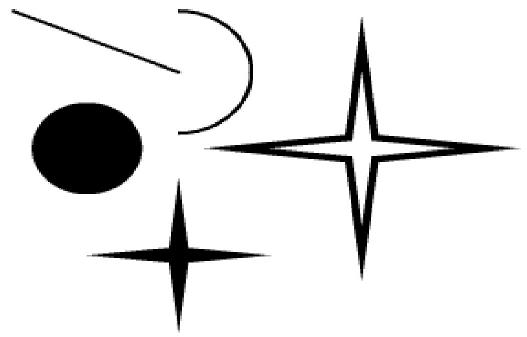
Antialiasing

- 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.

```
#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

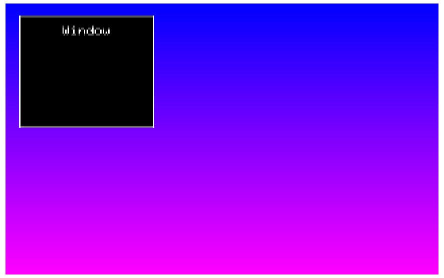
- **Management system for a hierarchic window structure**
  Each layer has its own desktop window. Each desktop window can have its own hierarchical tree of child windows.
- **Callback mechanism based system**
  Communication is based on an event driven callback mechanism. All drawing operations should 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

```
#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);
  }
}

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);
  }
}
```

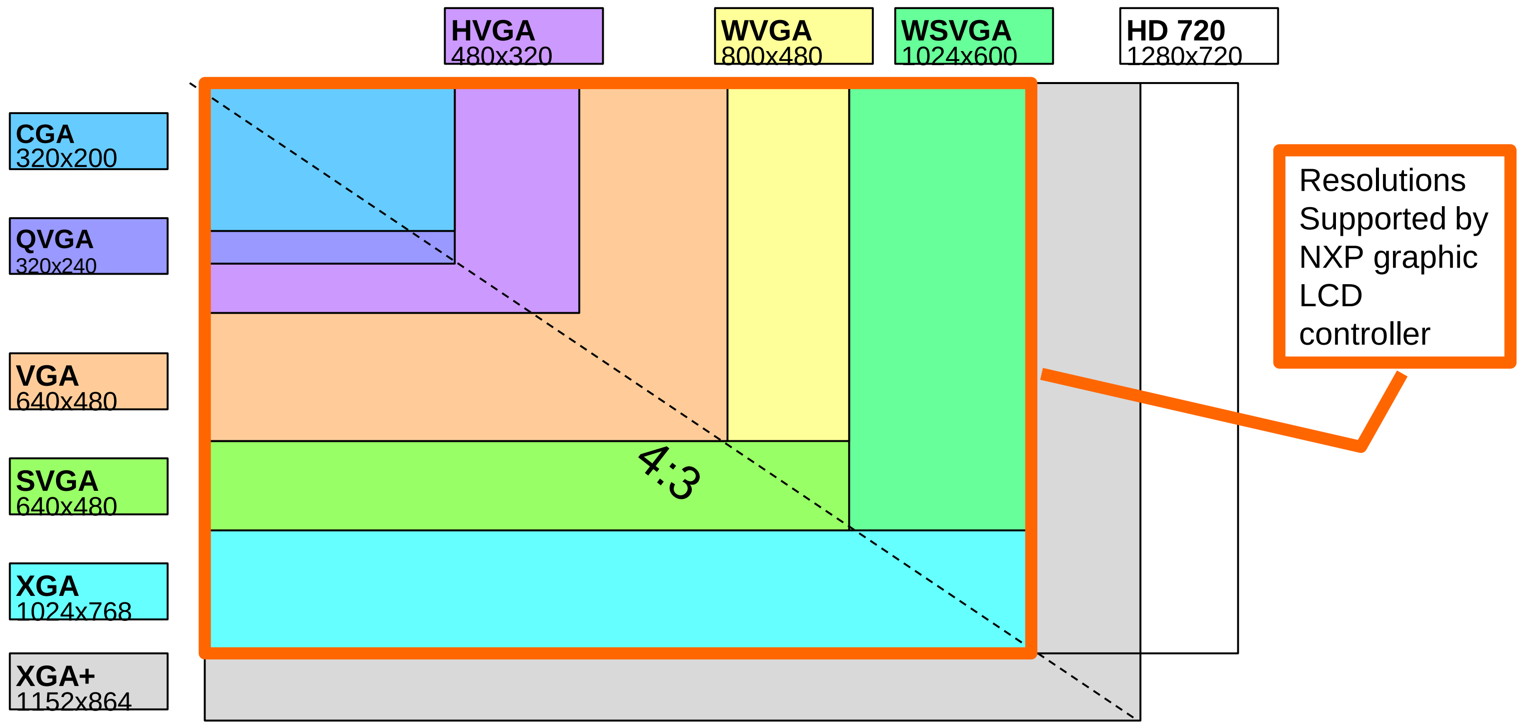


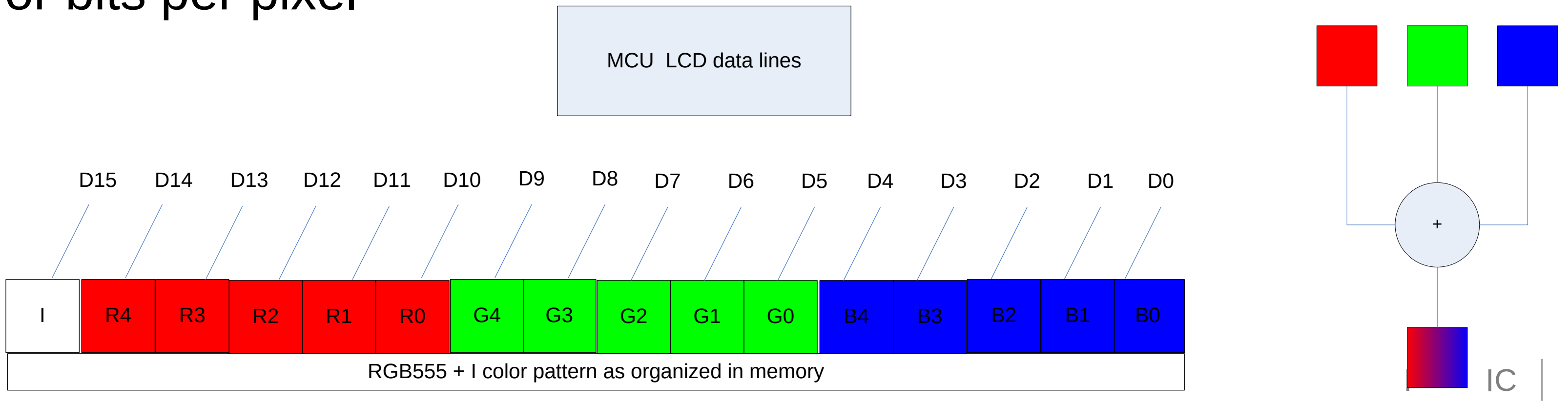
Supplemental Information

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

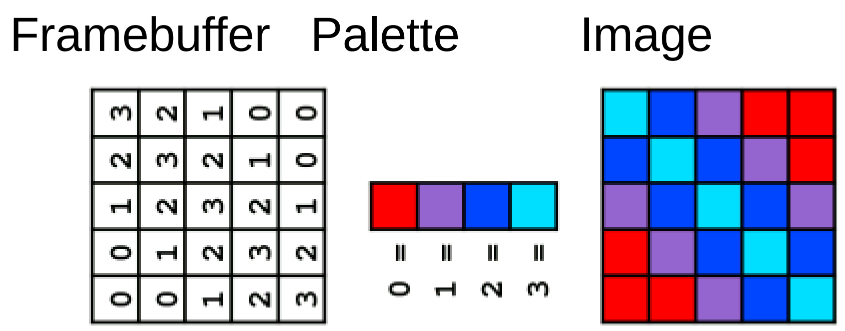
- Resolution x Color Depth = Total bits needed (divide by 8 for bytes)
- Framebuffer = Memory buffer containing complete frame (bitmap) of data

Resolution		1 bits/ pixel	2 bits/ pixel	4 bits/ pixel	8 bits/ pixel	16 bits/ pixel	24 bits/ pixel
XGA	1024x768	98,304	196,608	393,216	786,432	1,572,864	2,359,296
WVGA	800x480	48,000	96,000	192,000	384,000	768,000	1,152,000
VGA	640x480	38,400	76,800	153,600	307,200	614,400	921,600
WQVGA	480x272	16,320	32,640	65,280	130,560	261,120	391,680
QVGA	320x240	9,600	19,200	38,400	76,800	153,600	230,400
CGA	320x200	8,000	16,000	32,000	64,000	128,000	192,000

Example: (480 x 272) x 16bpp = 2,088,960 bits
To convert to bytes : 2088960 ÷ 8 = 261,120 bytes

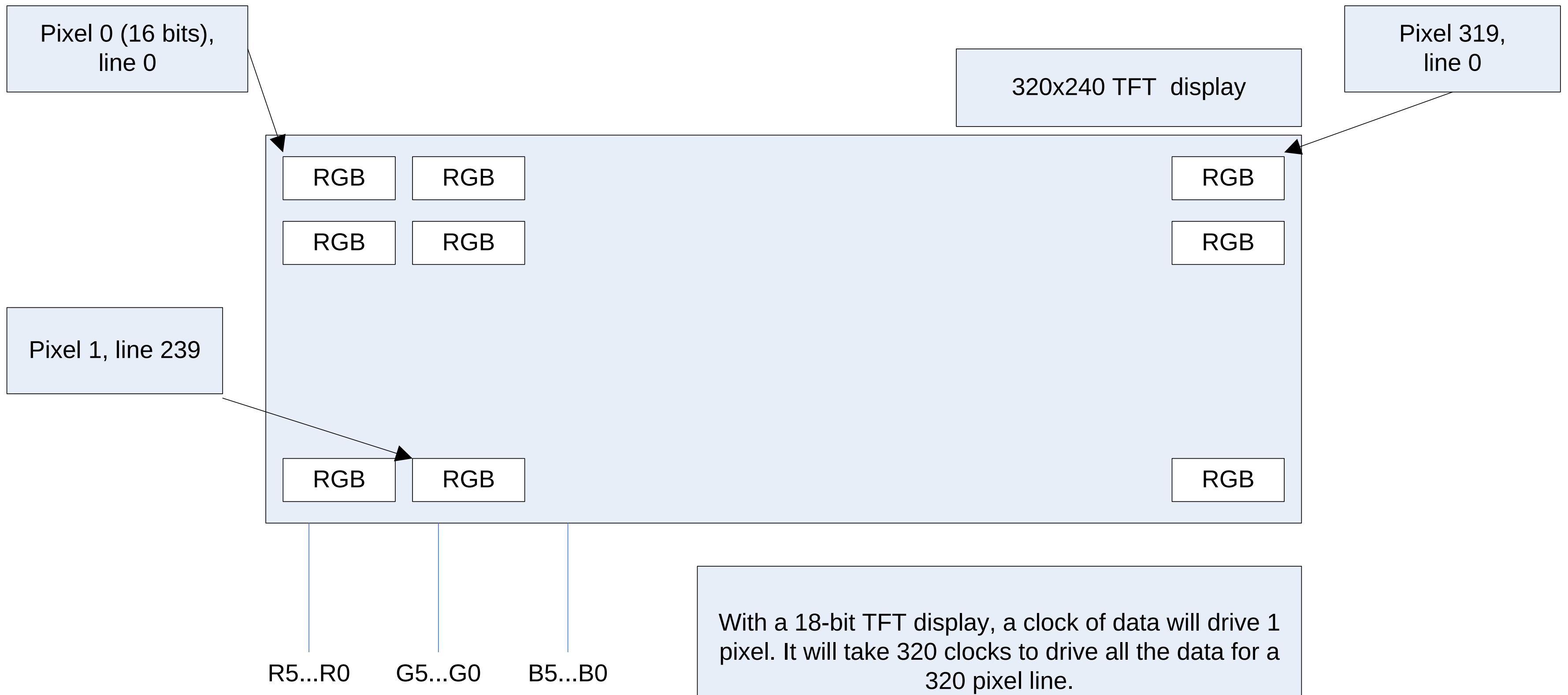
Palette Based Frame Buffer

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



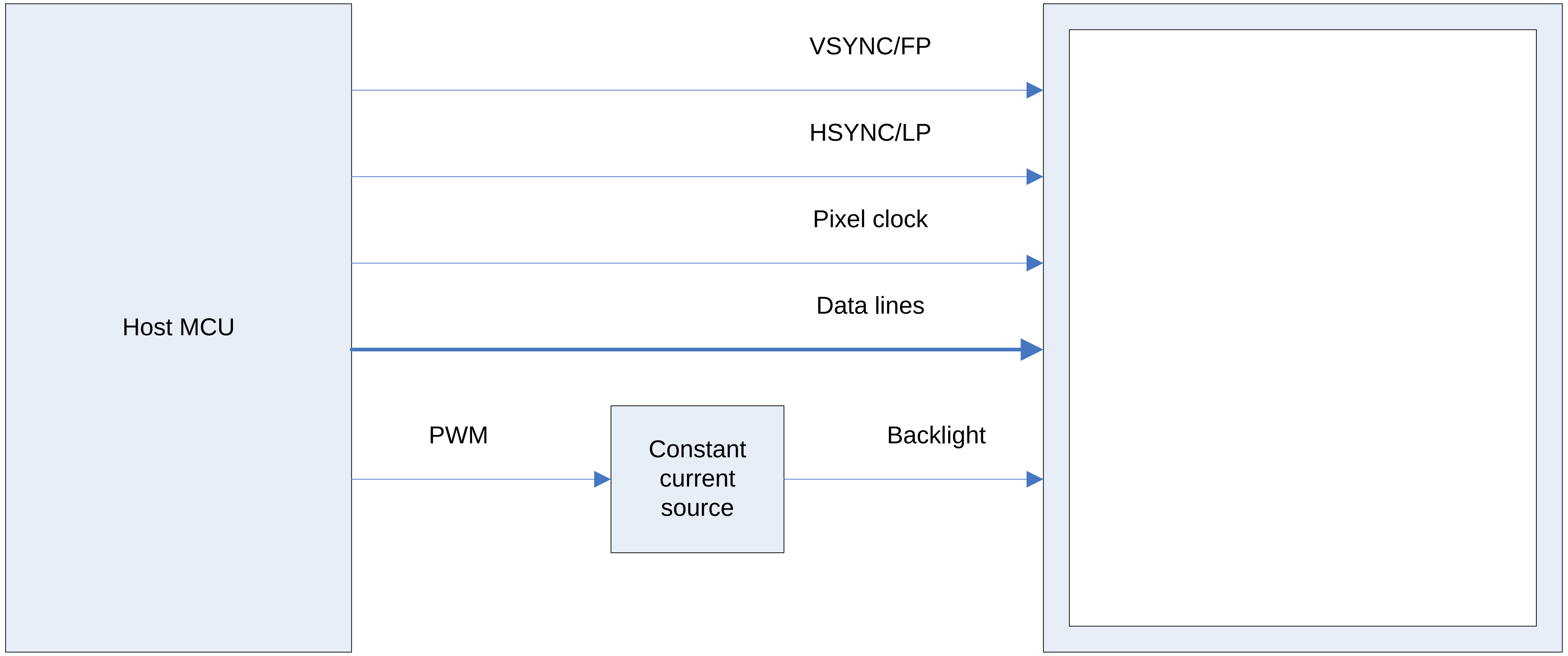
- NXP microcontrollers have 256 entries to support
  – 1, 2, 4, or 8 bpp palletized color displays for color STN and TFT
  – 1, 2, or 4 bits-per-pixel (bpp) palletized displays for mono STN

LCD TFT Display Pixel Layout



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

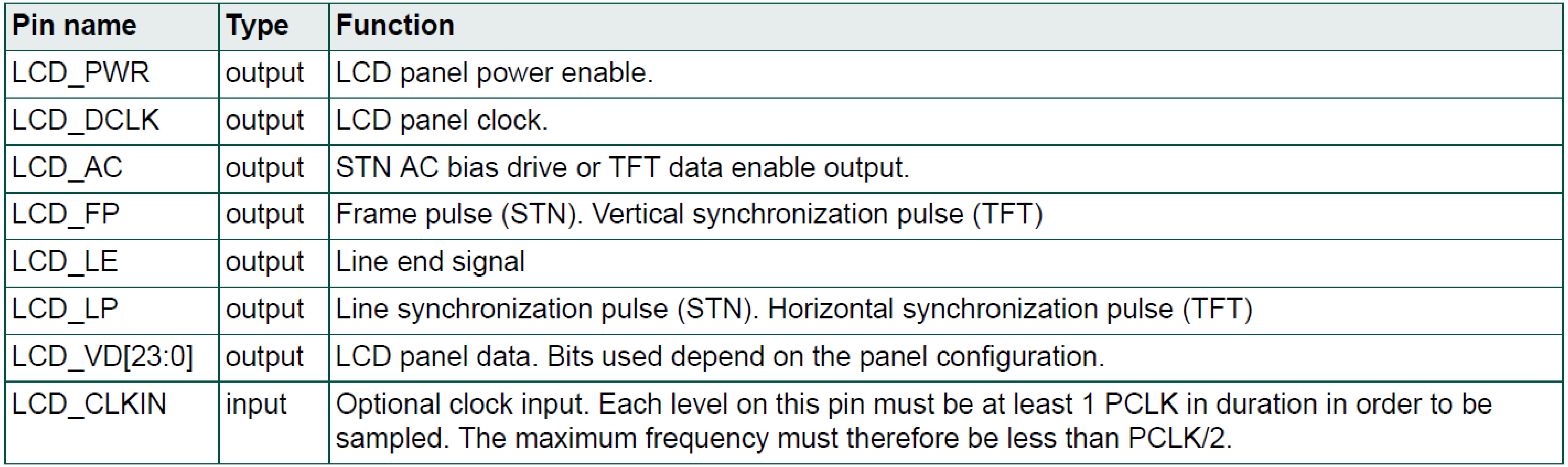
- $Refresh_Rate(Hz) = $
$$\frac{pixel\_clock\_rate}{[(vertical\_resolution + vertical\_front\_porch + vertical\_back\_porch) \times \ (pixel\_clocks\_per\_data\_line + horizontal\_front\_porch + horizontal\_back\_porch)]}$$

- 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

- $Refresh\_Rate = \frac{6500000}{[(240+5+1)\times(320+20+10)]} = 75.5\ Hz$

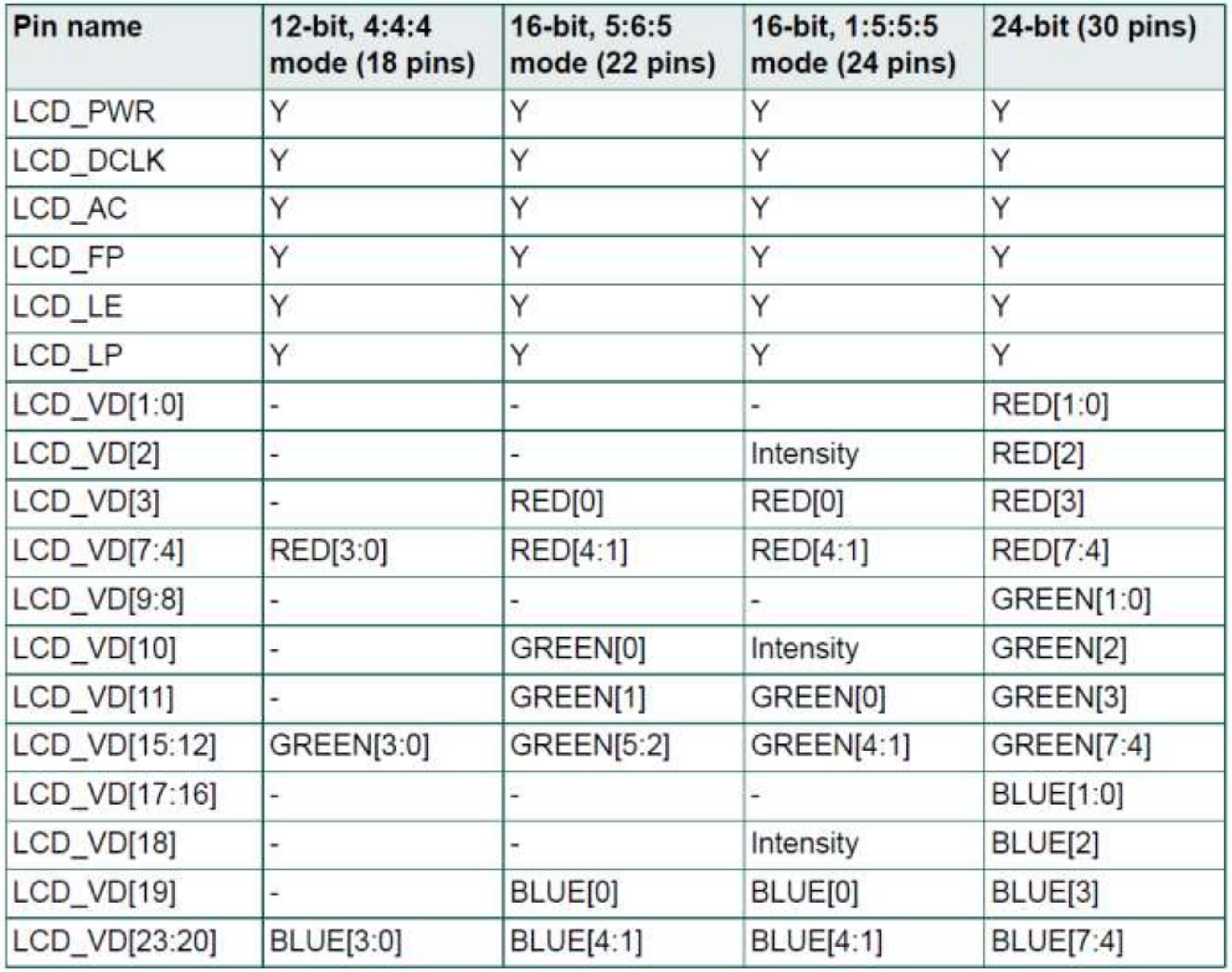
LCD Controller Signals

- 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.



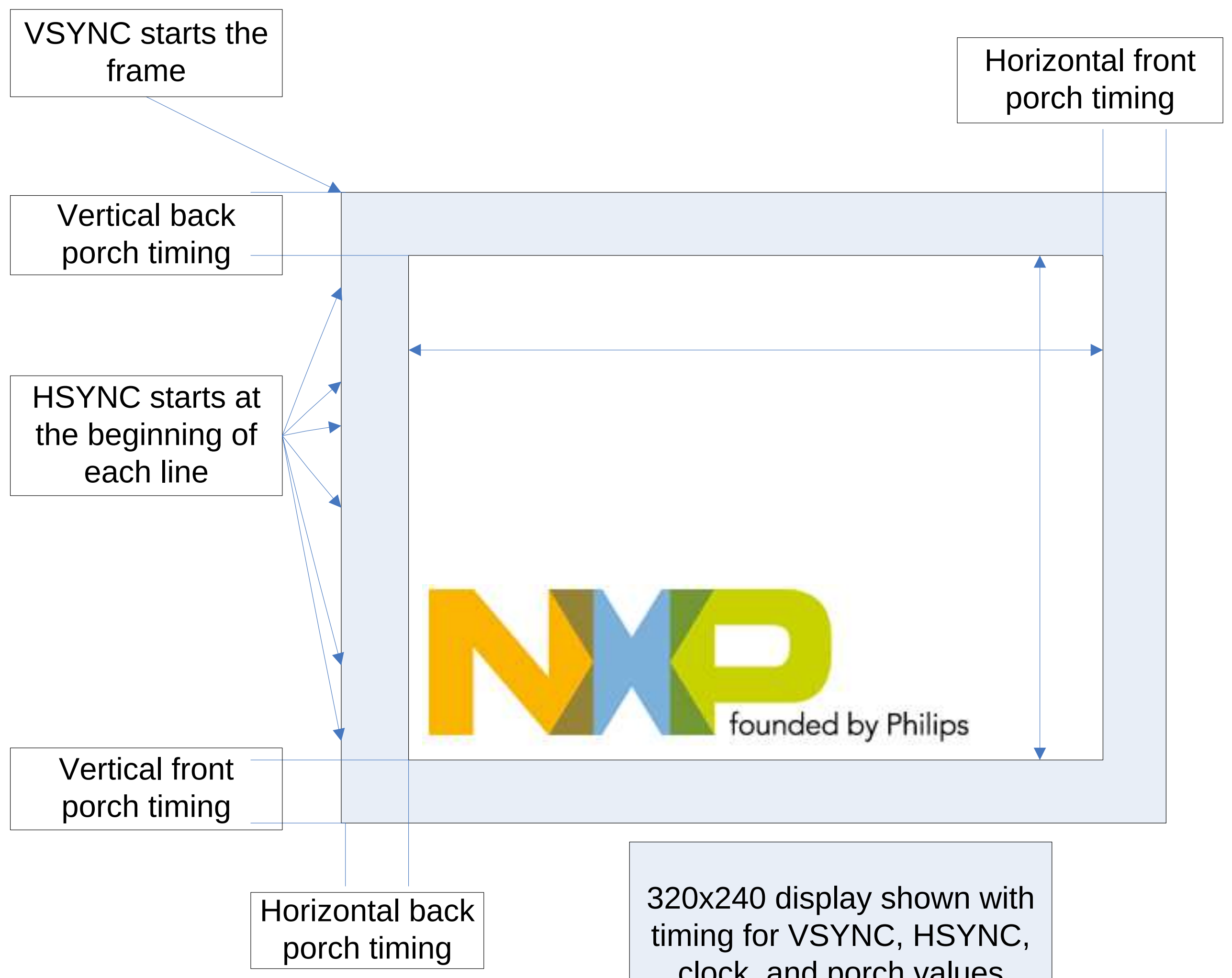
Pin name	Type	Function
LCD_PWR	output	LCD panel power enable.
LCD_DCLK	output	LCD panel clock.
LCD_AC	output	STN AC bias drive or TFT data enable output.
LCD_FP	output	Frame pulse (STN). Vertical synchronization pulse (TFT)
LCD_LE	output	Line end signal
LCD_LP	output	Line synchronization pulse (STN). Horizontal synchronization pulse (TFT)
LCD_VD[23:0]	output	LCD panel data. Bits used depend on the panel configuration.
LCD_CLKIN	input	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.

LCD TFT Signals



Pin name	12-bit, 4:4:4 mode (18 pins)	16-bit, 5:6:5 mode (22 pins)	16-bit, 1:5:5:5 mode (24 pins)	24-bit (30 pins)
LCD_PWR	Y	Y	Y	Y
LCD_DCLK	Y	Y	Y	Y
LCD_AC	Y	Y	Y	Y
LCD_FP	Y	Y	Y	Y
LCD_LE	Y	Y	Y	Y
LCD_LP	Y	Y	Y	Y
LCD_VD[1:0]	-	-	-	RED[1:0]
LCD_VD[2]	-	-	Intensity	RED[2]
LCD_VD[3]	-	RED[0]	RED[0]	RED[3]
LCD_VD[7:4]	RED[3:0]	RED[4:1]	RED[4:1]	RED[7:4]
LCD_VD[9:8]	-	-	-	GREEN[1:0]
LCD_VD[10]	-	GREEN[0]	Intensity	GREEN[2]
LCD_VD[11]	-	GREEN[1]	GREEN[0]	GREEN[3]
LCD_VD[15:12]	GREEN[3:0]	GREEN[5:2]	GREEN[4:1]	GREEN[7:4]
LCD_VD[17:16]	-	-	-	BLUE[1:0]
LCD_VD[18]	-	-	Intensity	BLUE[2]
LCD_VD[19]	-	BLUE[0]	BLUE[0]	BLUE[3]
LCD_VD[23:20]	BLUE[3:0]	BLUE[4:1]	BLUE[4:1]	BLUE[7:4]

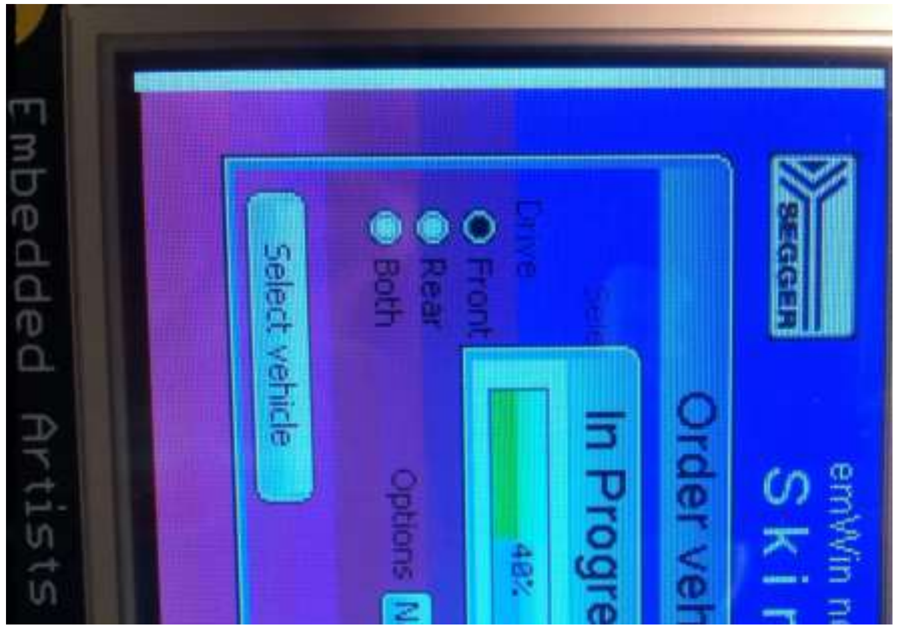
Driving the LCD – Various Timings

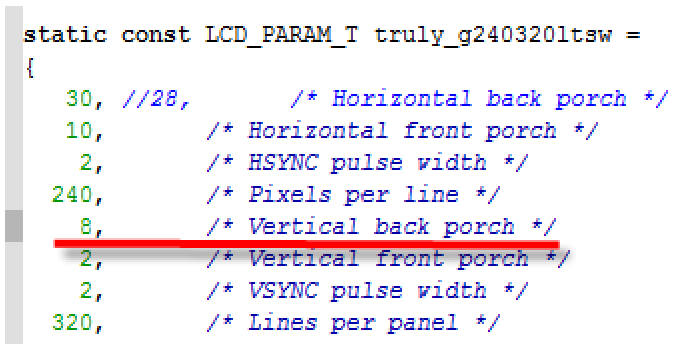

VSYNC starts the frame
Horizontal front porch timing
Vertical back porch timing
HSYNC starts at the beginning of each line
founded by Philips
Vertical front porch timing
Horizontal back porch timing

320x240 display shown with timing for VSYNC, HSYNC, clock, and porch values

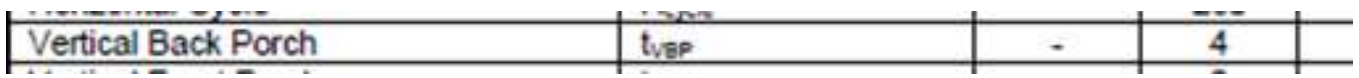
Snapshot of incorrect LCD settings

- 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.





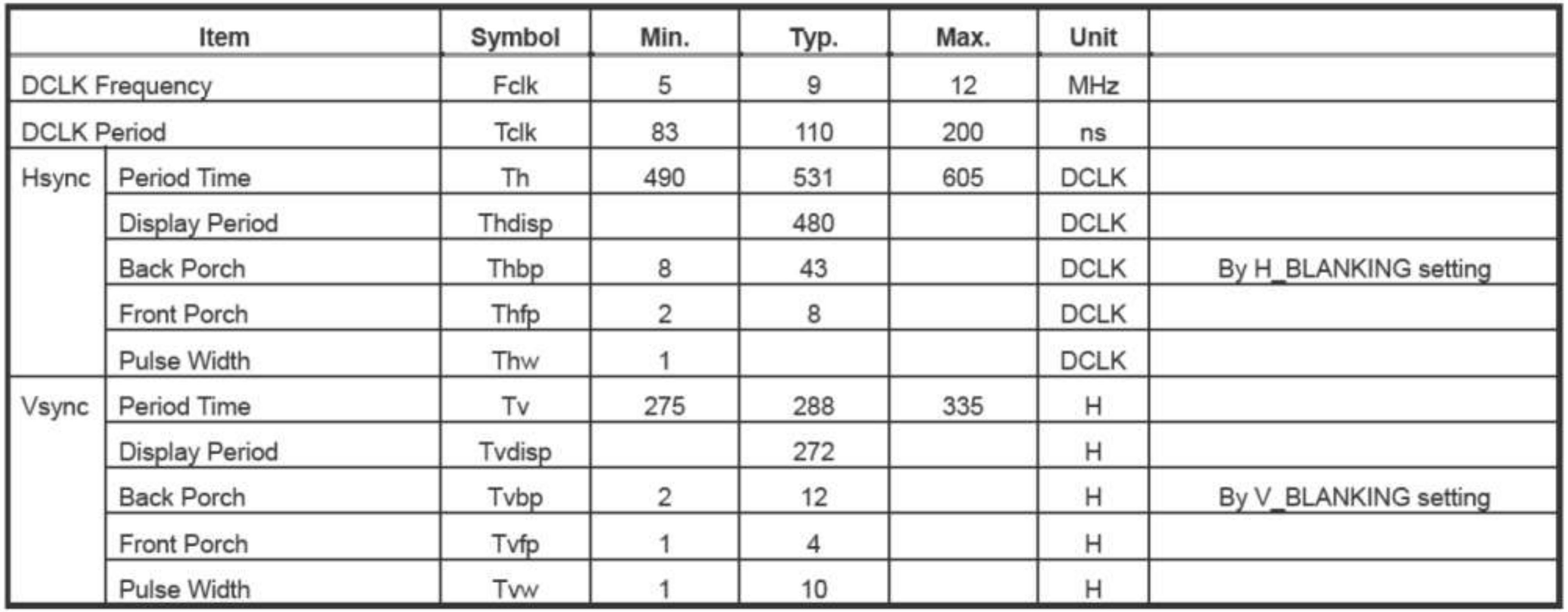
```
static const LCD_PARAM_T truly_g240320ltsw =
{
   30, //28,       /* Horizontal back porch */
   10,        /* Horizontal front porch */
    2,        /* HSYNC pulse width */
  240,        /* Pixels per line */
    8,        /* Vertical back porch */
    2,        /* Vertical front porch */
    2,        /* VSYNC pulse width */
  320,        /* Lines per panel */
```



Vertical Back Porch	t_{VBP}		-	4

Example : 480 x 272 LCD TFT Display Timing Table

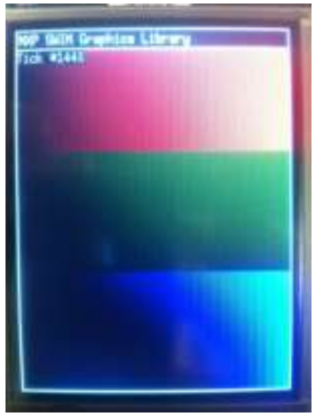
- Below is the timing table of LCD used in LPCXpresso54608 Development board.

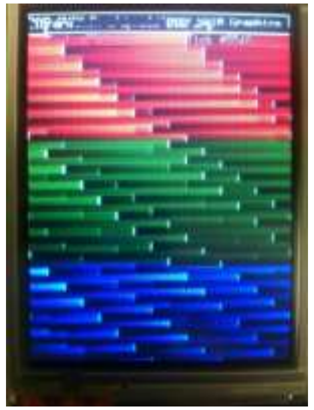


Item		Symbol	Min.	Typ.	Max.	Unit	
DCLK Frequency		Fclk	5	9	12	MHz	
DCLK Period		Tclk	83	110	200	ns	
Hsync	Period Time	Th	490	531	605	DCLK	
	Display Period	Thdisp		480		DCLK	
	Back Porch	Thbp	8	43		DCLK	By H_BLANKING setting
	Front Porch	Thfp	2	8		DCLK	
	Pulse Width	Thw	1			DCLK	
Vsync	Period Time	Tv	275	288	335	H	
	Display Period	Tvdisp		272		H	
	Back Porch	Tvbp	2	12		H	By V_BLANKING setting
	Front Porch	Tvfp	1	4		H	
	Pulse Width	Tvw	1	10		H	

LCD Tearing

- Tearing:

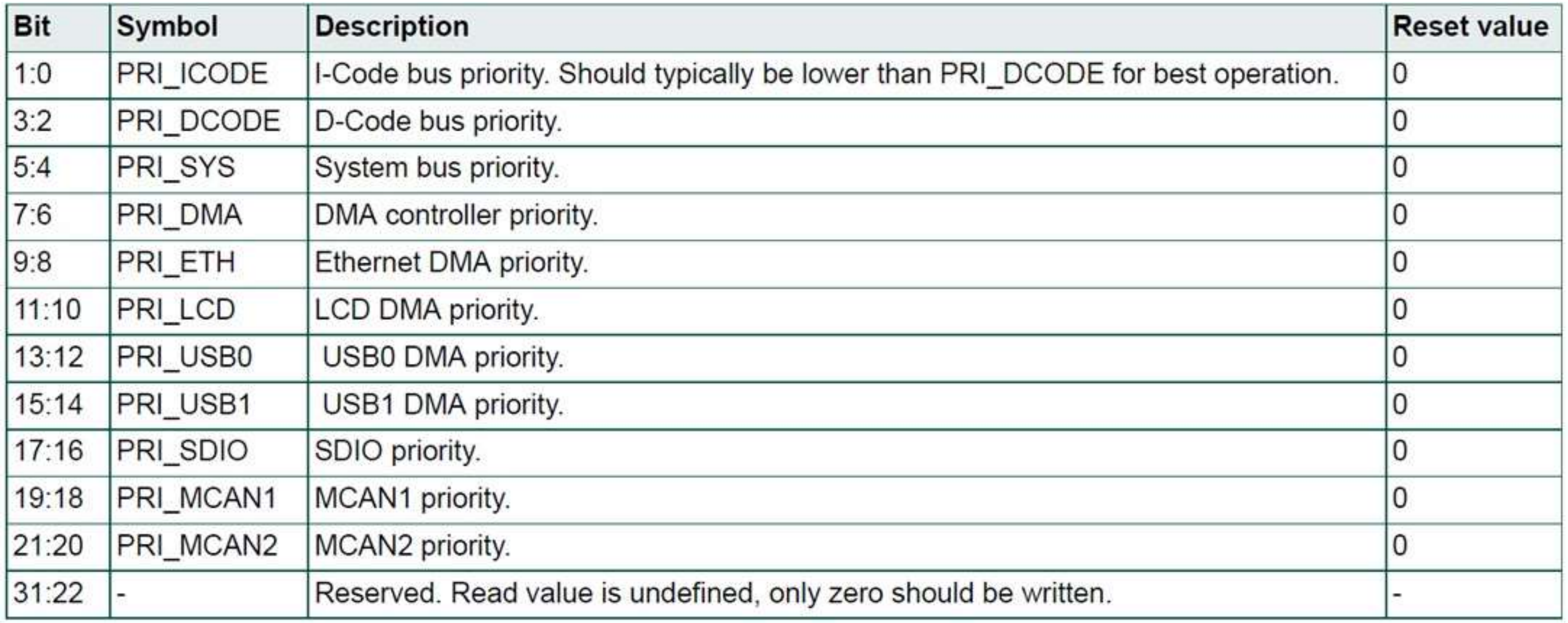




- Result of LCD DMA unable to service the LCD FIFO in time
- Use the FIFO Underflow to monitor for this
- Workarounds
- Change AHB priority – next slide
- Slow down frame refresh rate, pixel clock if possible
- Use 32-bit wide external memories
- Increase the SDRAM clock speed, use faster SRAM
- Profile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

LPC54608 LCD AHB Priority

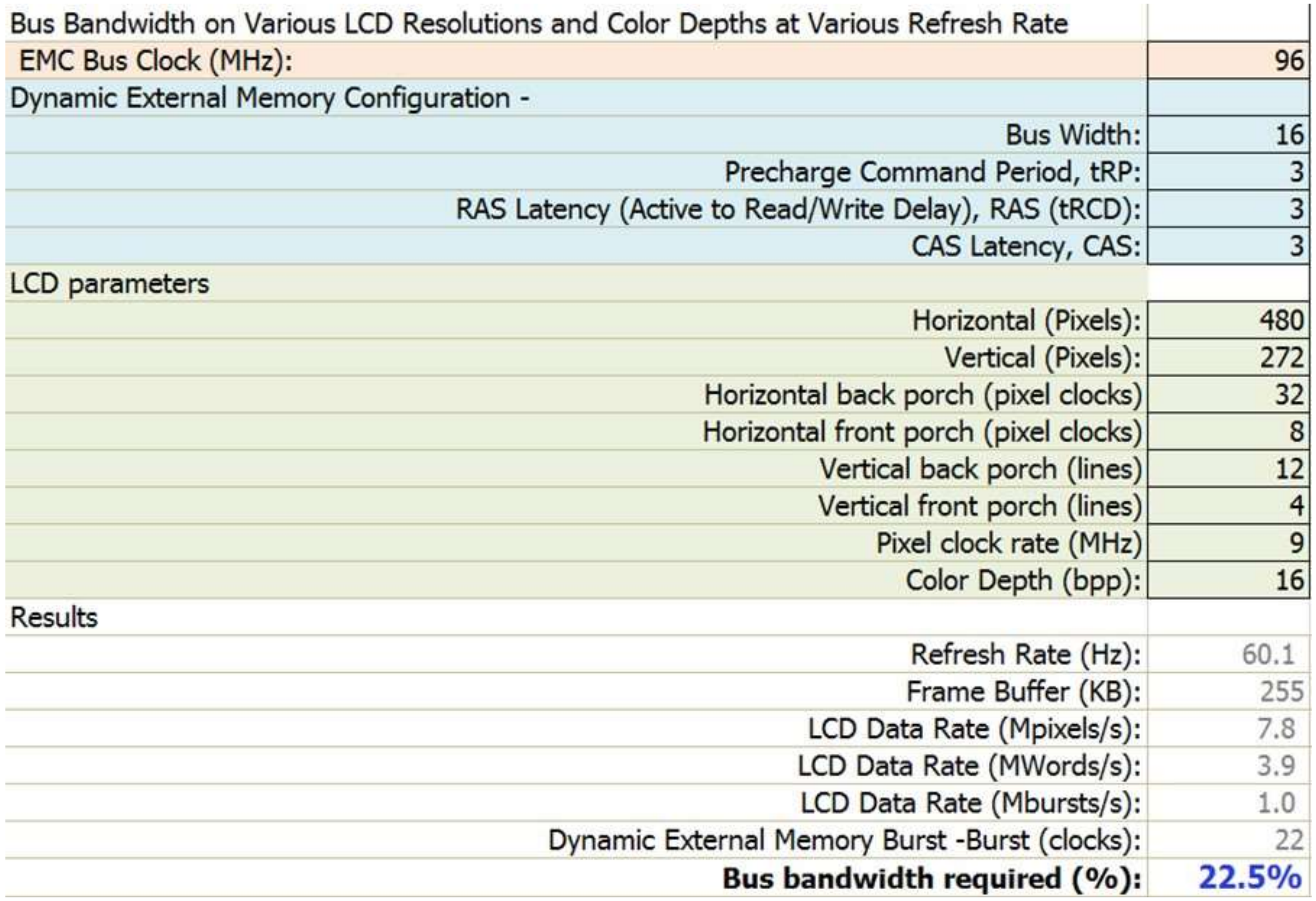
- AHB Matrix Priority Register (AHBMATPRIO - 0x4000 0010)
- The values used for the various priorities are 3 = highest, 0 = lowest



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

- To give priority to the LCD DMA use the value 0x0000 0C09

Bus Bandwidth Calculator



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

SECURE CONNECTIONS
FOR A SMARTER WORLD