

# ADDING CONNECTIVITY AND GRAPHICS CAPABILITY TO LPC MCUS

FTF-INS-N1970

ALLEN WILLSON, PRODUCT MANAGER  
BRENDON SLADE, DIRECTOR, LPC  
TOOLS & ECOSYSTEM  
MAY 16, 2016



PUBLIC








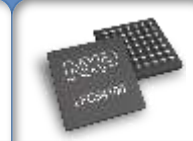


SECURE CONNECTIONS  
FOR A SMARTER WORLD


# AGENDA

- Session Objectives
- Development Platform Overview
- LPC4088 Device and Embedded Artists Display Module
- Graphics Solutions
  - Application of Graphic Displays
  - Choosing a Graphics Package
  - emWIN Technical Session, including USB connectivity




# NXP LPC Microcontroller Portfolio At-a-Glance

From entry level			To high performance				
<p>Easy to use Exceptional power efficiency Lowest pin count</p>			<p>Best power efficiency Advanced connectivity Flexible peripherals</p>				
							
<b>LPC800 Series</b>	<b>LPC1100 Series</b>	<b>LPC1200 Series</b>	<b>LPC1300 Series</b>	<b>LPC1500 Series</b>	<b>LPC54100 Series</b>	<b>LPC1700 Series</b>	<b>LPC4000 Series</b>
<p><b>Low power, basic control and connectivity</b></p> <ul style="list-style-type: none"> <li>• 30 MHz Cortex-M0+ core</li> <li>• Basic serial connectivity</li> <li>• Basic analog</li> <li>• Low-pincount packages including TSSOP and HVQFN and XSON</li> <li>• Ideal for 8-/16-bit transition</li> </ul>	<p><b>Power efficient, broad selection, industry-standard connectivity</b></p> <ul style="list-style-type: none"> <li>• 50 MHz Cortex-M0+ &amp; M0 cores</li> <li>• Serial connectivity: USB with PHY, CAN with transceiver</li> <li>• Best-in-class analog</li> <li>• Broad package selection</li> <li>• Migration path to LPC1300 Series</li> </ul>	<p><b>Noise immunity for industrial applications</b></p> <ul style="list-style-type: none"> <li>• 45 MHz Cortex-M0 core</li> <li>• High-immunity rating (IEC61697-1)</li> <li>• 8 kV ESD protection</li> <li>• Basic analog</li> <li>• Real-time clock</li> <li>• Fm I<sup>2</sup>C with 10x bus-drive capability</li> </ul>	<p><b>Performance and basic connectivity</b></p> <ul style="list-style-type: none"> <li>• Up to 72 MHz Cortex-M3 core</li> <li>• Serial connectivity: USB, CAN</li> <li>• Pin-compatible upgrade for most LPC1100 Series devices</li> </ul>	<p><b>High-precision motion control</b></p> <ul style="list-style-type: none"> <li>• Up to 72 MHz Cortex-M3 core</li> <li>• Optimized for sensed &amp; sensorless brushless motor control; free FOC firmware</li> <li>• Serial connectivity: USB, CAN</li> <li>• Advanced analog subsystem and SCTimer/PWM</li> </ul>	<p><b>Ultra-low-power for always-on sensor processing</b></p> <ul style="list-style-type: none"> <li>• Up to 100 MHz single- &amp; dual-core: Cortex-M4F &amp; M0+ (opt.)</li> <li>• Optimized for sensor listening, aggregation, fusion, and communication</li> <li>• Ultra-low 'power down' mode, down to 3 µA for sensor listening</li> <li>• Scalable power performance</li> </ul>	<p><b>High performance with DSP options, multi-connectivity, advanced peripherals</b></p> <ul style="list-style-type: none"> <li>• Up to 120 MHz Cortex-M3 core</li> <li>• Advanced connectivity: USB, CAN, Ethernet</li> <li>• Graphic LCD controller</li> <li>• Pin-compatible migration path to LPC4000 Series and ARM7 LPC2x00 Series</li> </ul>	<p><b>Best performance with DSP and dual-core options, multi-high-speed connectivity, advanced peripherals</b></p> <ul style="list-style-type: none"> <li>• Industry's highest-performing Cortex-M3 core, up to 180 MHz</li> <li>• LPC18Sxx family includes integrated security features</li> <li>• Advanced conn.: dual Hi-Speed USB, dual CAN, 10/100 Ethernet</li> <li>• Advanced, flexible timers for event-driven timing and PWM applications</li> <li>• Drop-in compatible with LPC4300 Series</li> </ul>
<a href="#">LEARN MORE</a>	<a href="#">LEARN MORE</a>	<a href="#">LEARN MORE</a>	<a href="#">LEARN MORE</a>	<a href="#">LEARN MORE</a>	<a href="#">LEARN MORE</a>	<a href="#">LEARN MORE</a>	<a href="#">LEARN MORE</a>



LPC1800 Series  
LPC18Sxx Family



LPC4300 Series  
LPC43Sxx Family

**Best performance with DSP and dual-core options, multi-high-speed connectivity, advanced peripherals**

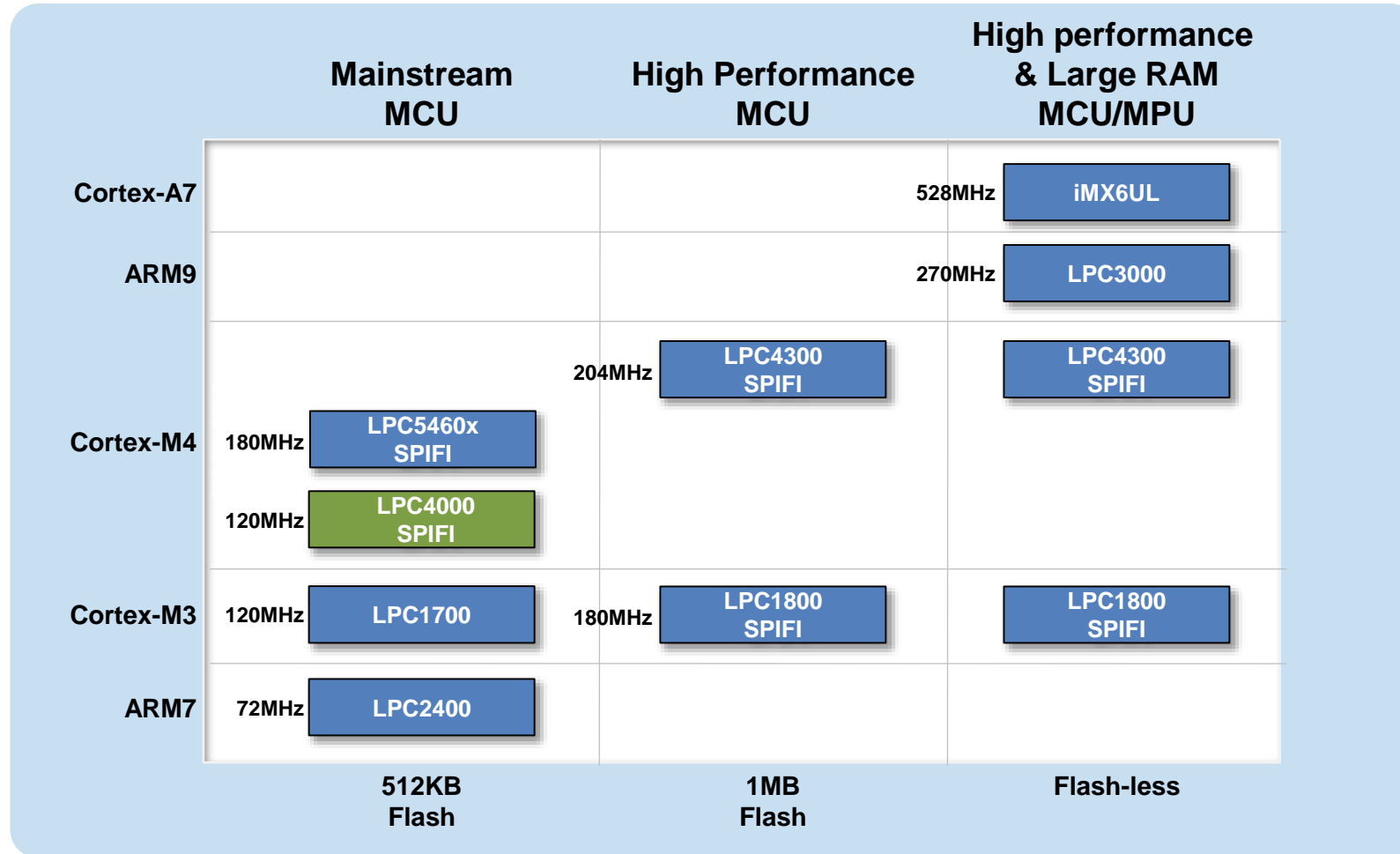
- Industry's highest-performing Cortex-M3 core, up to 180 MHz
- LPC18Sxx family includes integrated security features
- Advanced conn.: dual Hi-Speed USB, dual CAN, 10/100 Ethernet
- Advanced, flexible timers for event-driven timing and PWM applications
- Drop-in compatible with LPC4300 Series

- Up to 204 MHz Cortex-M4F core with DSP capabilities and Cortex-M0 coprocessor(s)
- LPC43Sxx family includes integrated security features
- Partition tasks across cores to optimize performance
- Advanced conn.: dual Hi-Speed USB, dual CAN, 10/100 Ethernet, configurable high-speed serial I/O
- Best-in-class analog, up to 80 Msps, 12-bit ADC



# Portfolio Breadth & Scalability

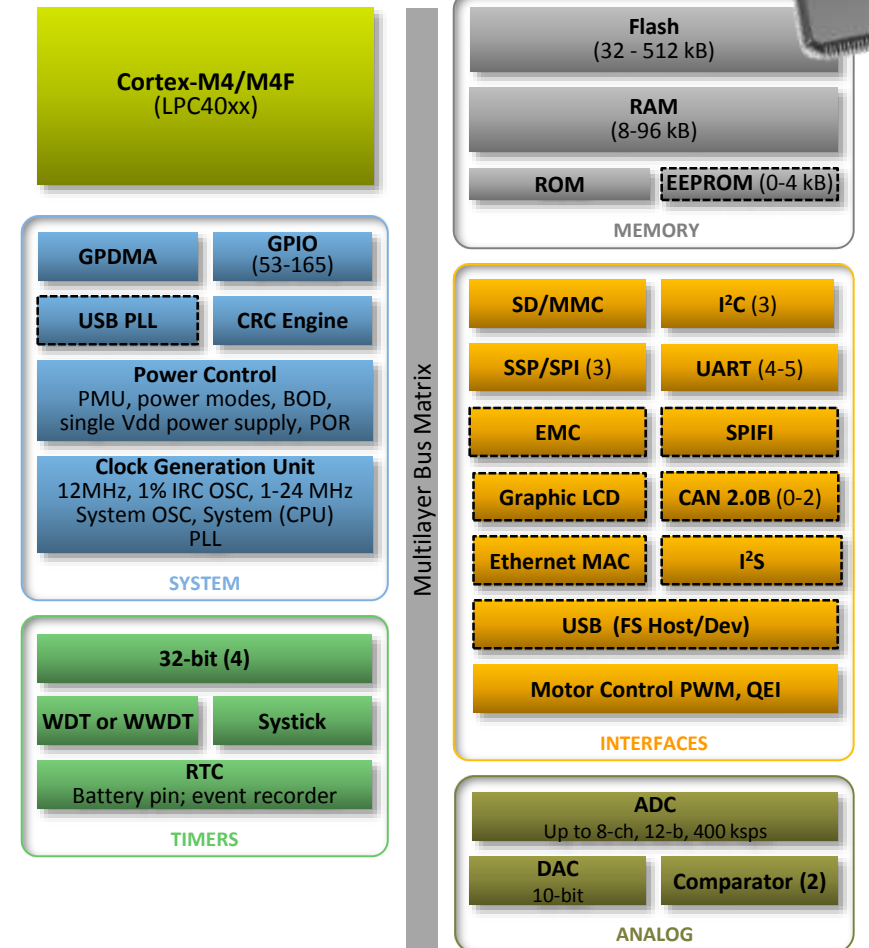
Driving Displays with More than 40 parts & 10 years of market experience



# 4000 Series

## High performance, multi-connectivity, advanced peripherals

- 120 MHz Cortex-M4/M4F (LPC40xx)
  - 512 kB Flash; Up to 96 KB RAM
  - XIP from QSPI via SPIFI
- Wide range of advanced connectivity
  - Full Speed USB with on-chip PHY and certified drivers
  - Dual FS USB host capable
  - Graphic LCD supporting resolutions up to 1024 x 768
  - CAN 2.0B
  - 10/100 Ethernet
- Pin compatibility
  - LPC17xx with ARM7 LPC2x00 and LPC40xx
  - LPC40xx drop-in compatible with LPC177x/8x and ARM7 LPC2x00



# Graphics LCD Controller

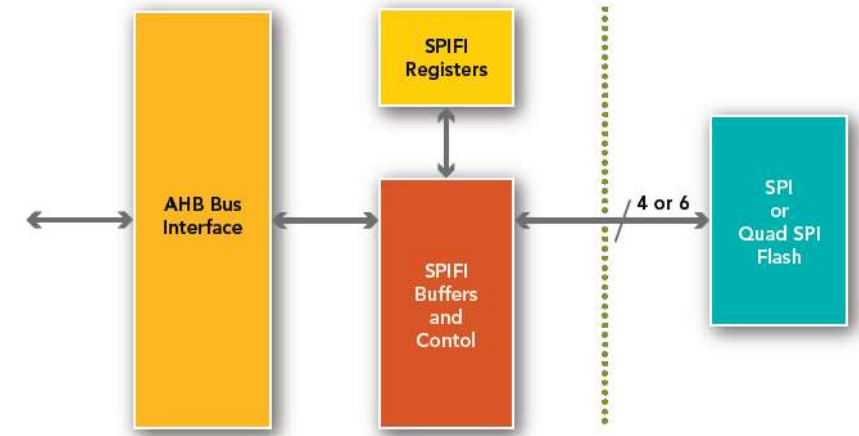
- Key features
  - Support for STN and TFT panels
  - **Up to 1024x768 resolution**
  - 24-bit LCD interface supports 24bpp (16M colors)
  - Palette table allows display of up to 256 of 64K colors
  - Adjustable LCD bus size supports various panel bus configurations
  - Dedicated LCD DMA controller
  - 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 and ExpressLogic's GUIX
- LPCOpen and Board Support Packages
  - Significantly reduces your software porting efforts
  - Porting guide available for non-standard LCDs



# Memory

## SPIFI (SPI Flash Interface)

- Enables Flash to appear in MCU memory map and be read like other on-chip memory
- Why use SPIFI?
  - Cost. Use small, inexpensive serial Flash in place of larger, more expensive parallel Flash
  - Performance. Approaches internal Flash performance (~70%)
  - Space. Saves board space and pins (NOR v Q-SPI Flash)
  - App size. **Ideal for storing image/data**, freeing internal Flash for app use
- And SPIFI supports
  - Multiple Q-SPI vendors
  - Code execution and data access and booting
  - DMA





# Embedded Artists LPC4088 Display Module

- OEM-ready display module based on LPC4088 MCU
  - CE certified, ISO9001 produced
  - -20 to 60 degrees C operation (limited by LCD)
- 16MB Quad SPI flash and 32MB SDRAM
- 4.3" TFT LCD with projected capacitive touch panel
- Built-in CMSIS-DAP debug probe, option for external probe
- USB host and device connectors
- uSD/transflash memory card interface connector
- XBee compatible RF module connector
- mbed compatible, with several examples available
- LPCOpen driver/example package available from Embedded Artists





# APPLICATION OF DISPLAYS



# Graphic versus Segment LCD



7seg    15seg

## Segment LCDs

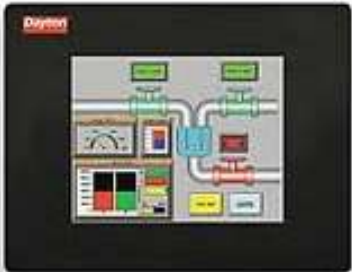
Advantages of segment-driven LCD displays are low communication overhead, low power, and virtually limitless but fixed display configurations.



## Graphic LCDs

Preferred over the character LCDs for applications where both character and graphical representation are required.

# Example Applications for Graphic LCDs

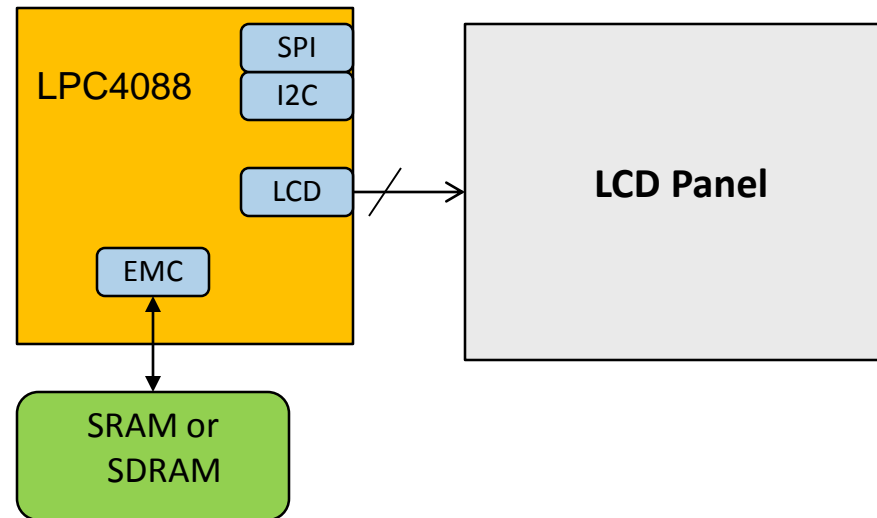
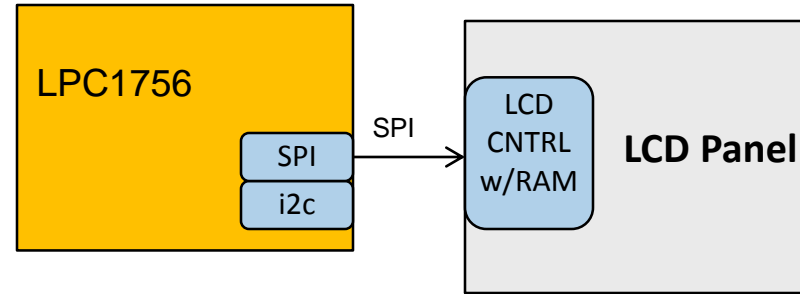


- Embedded applications for vibrant displays.
  - Home Automation and Security
    - Thermostats, security panel, intercom
  - Secure Transactions
    - POS Systems, Access Control, Ticketing
  - White Goods
    - High end Display and Human Interface
  - Industrial Human Machine Interface/ Programmable Logic Controls
    - RPM monitor, temp monitor, alarms
  - Medical Systems
    - Portable meters, large monitoring equipment
- Typical resolutions from CGA (320x200) to XGA (1024x768) and <15fps

# MCU with external or internal LCD controller

## Example interfaces MCU to LCD

- Serial interfaces to reduce pin count to the LCD
  - Lower resolution due to limited SPI bandwidth
  
- MCU with parallel LCD controller onchip
  - Can support mid-range resolutions.



# CHOOSING A GRAPHICS PACKAGE



# Challenges

- USER Expectations on the rise
- Management Expectations
- UI Development Time Consuming
- Many Developers do not have UI / Graphics experience
- TFT Displays
  - Present where they are not historically present
  - White Goods, Washing Machines, Ovens
- Quality and Competition
  - Visual Appearance and Screens are the first things people look at
  - Pressure to adapt

# Typical emWin HMI's





# 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)

# License terms



- Free to use with any current NXP ARM Cortex M0, M3 or M4 MCU
- No royalties or licensing fees when used with NXP MCUs
- No source (provided as a pre-compiled library)
- Source code available under license agreement from SEGGER



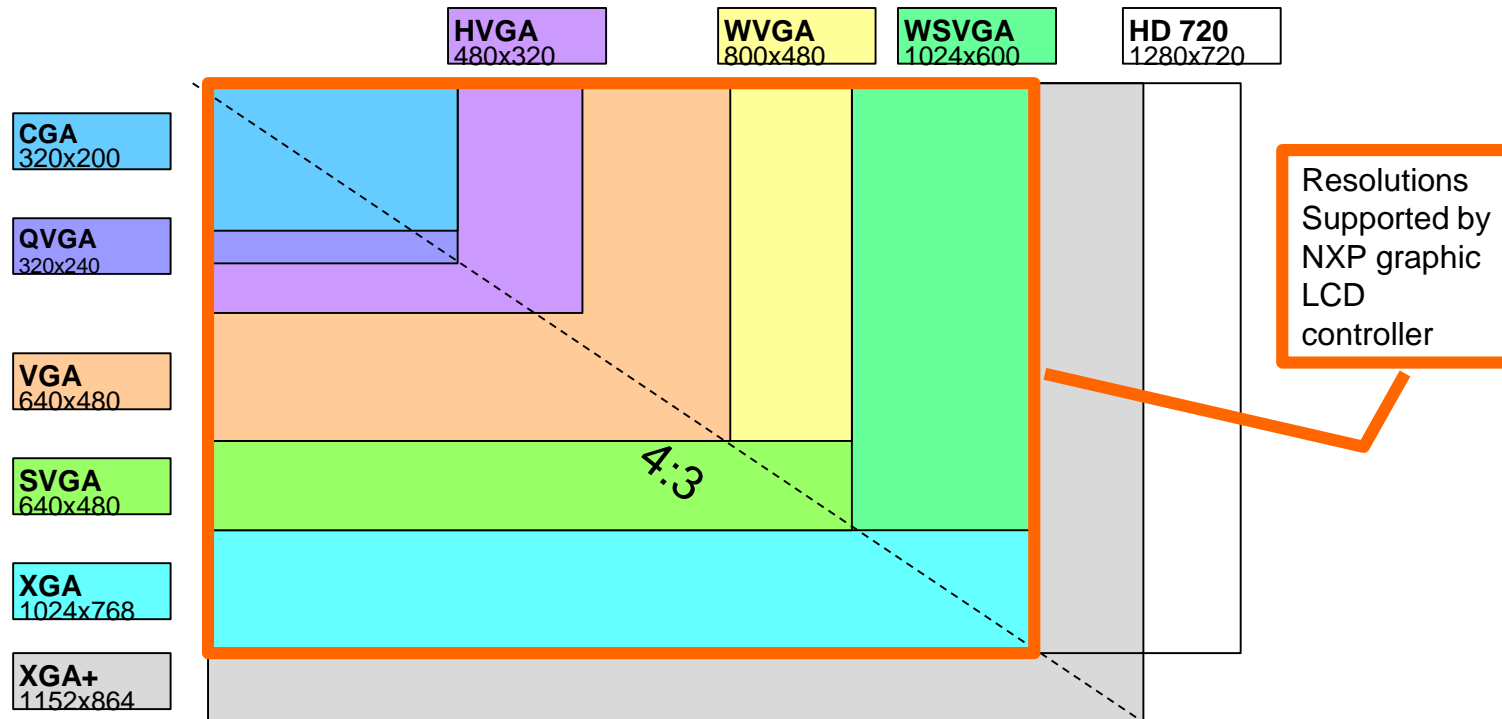
- The full license agreement is included in every installer, but there are essentially no limitations on the use of emWin with NXP MCUs
- The only restriction is that the emWin library is provided solely in object code ("library") format. Customers may use these libraries on NXP MCUs free of charge (without royalty or additional license fees), for both personal and commercial development
- As part of the licensing agreement with Segger, the source code for emWin can not be provided, but if you require the original source code for your own project, Segger offers special pricing for NXP customer's when upgrading from the NXP emWin library



# LCD CONTROLLER

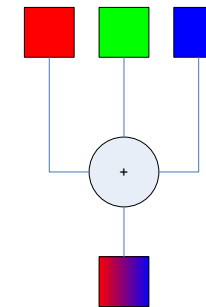
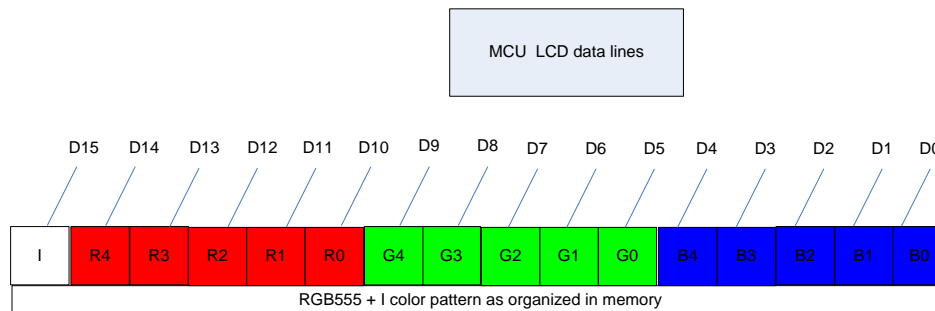


# Choosing an LCD: Resolution:



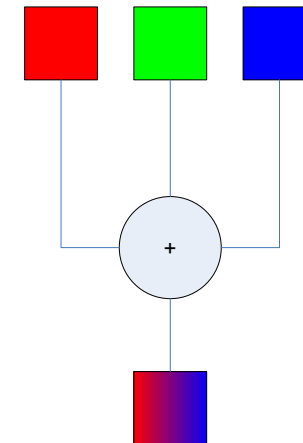
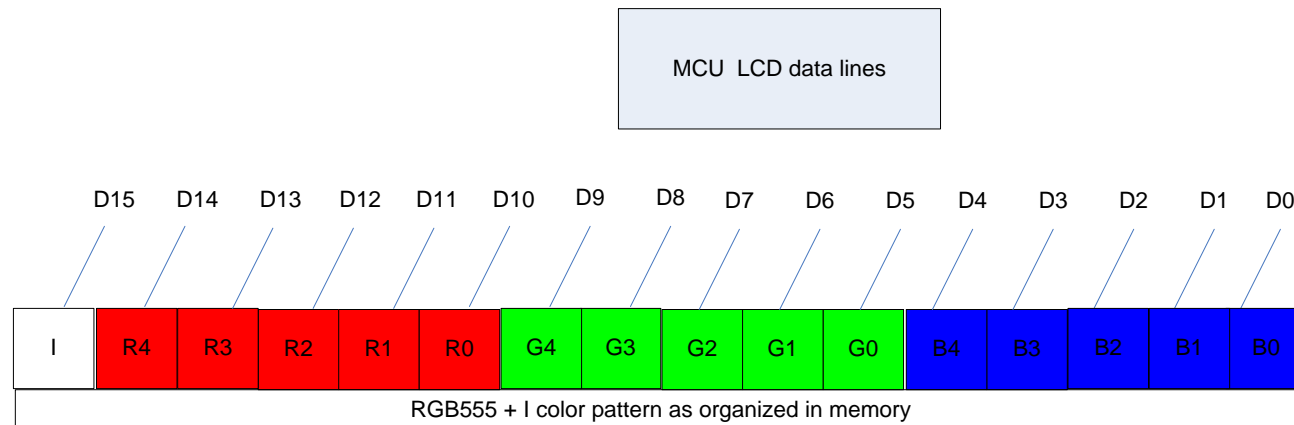
## Choosing an LCD: Color Depth

► Color depth or bits per pixel (bpp)



# Resolution and Color Depth

- Resolution is not measured in inches!
  - 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 x8 for bytes)
- Framebuffer = memory buffer containing complete frame (bitmap) of data

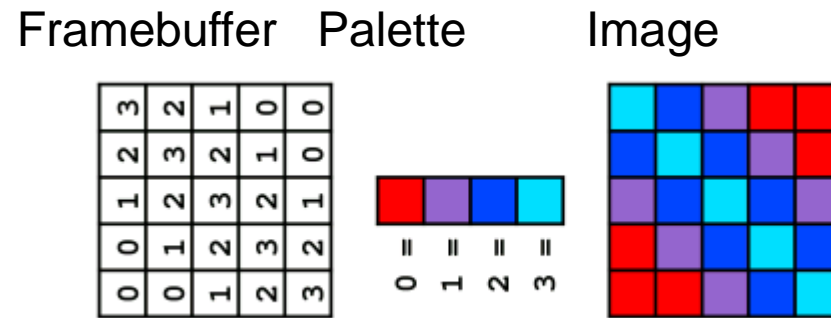
Resolution	1 bits/pixel	2 bits/pixel	4 bits/pixel	8 bits/pixel	16 bits/pixel	24 bits/pixel
XGA 1024x768	98,304	196,608	393,216	786,432	1,572,864	2,359,296
WVGA 800x480	48,000	96,000	192,000	384,000	768,000	1,152,000
VGA 640x480	38,400	76,800	153,600	307,200	614,400	921,600
WQVGA 480x272	16,320	32,640	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 x 8bits/byte = 261,120 bytes needed



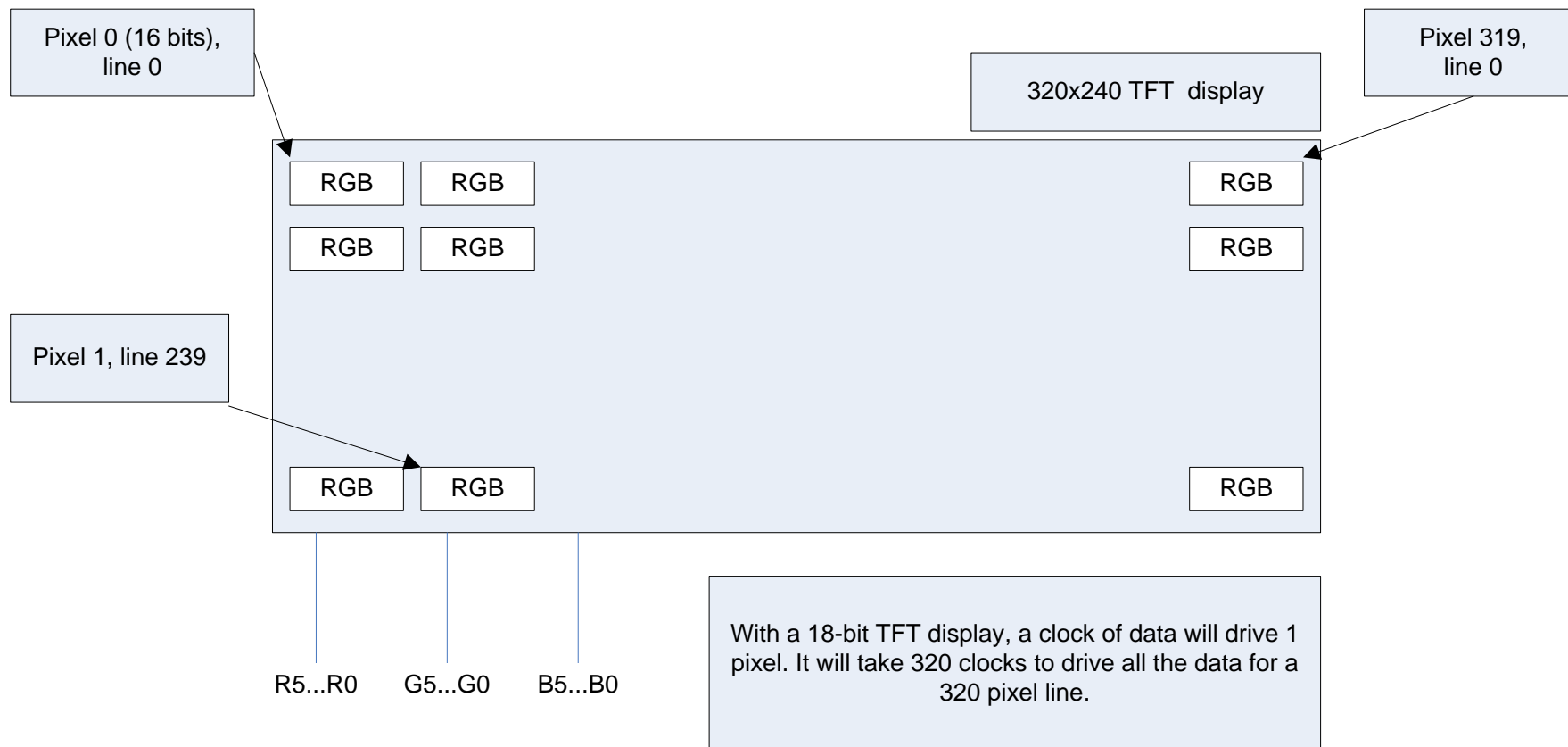
# Palette Based Frame Buffer

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

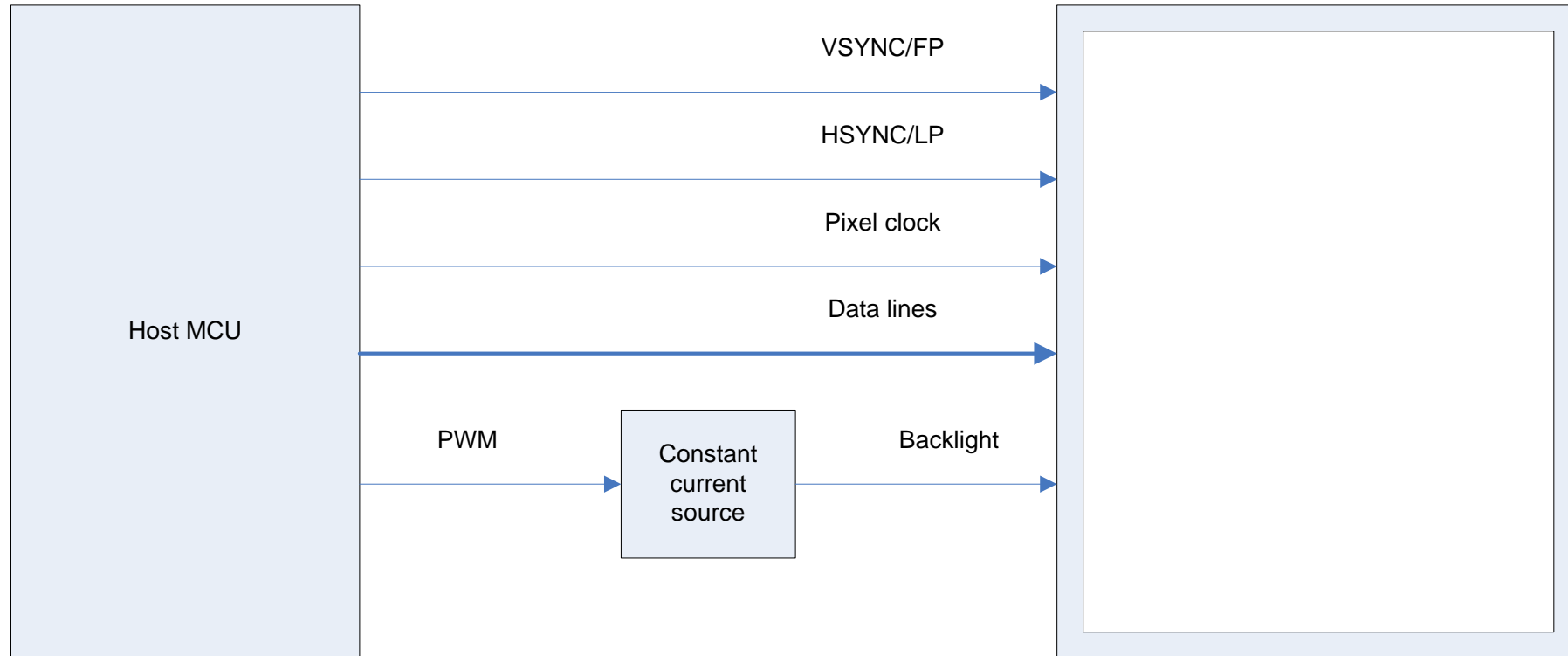


- NXP microcontrollers have 256 entries to support
  - ▶ 1, 2, 4, or 8 bpp palletized color displays for color STN and TFT
  - ▶ 1, 2, or 4 bits-per-pixel (bpp) palletized displays for mono STN

# LCD Clocked TFT



# Driving a clocked LCD bus



# Refresh Rate

- REFRESH\_RATE (Hz) =  
$$\text{pixel\_clock\_rate} / [(\text{vertical\_resolution} + \text{vertical\_front\_porch} + \text{vertical\_back\_porch}) * (\text{pixel\_clocks\_per\_data\_line} + \text{horizontal\_front\_porch} + \text{horizontal\_back\_porch})]$$
- Example :
  - 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 = 6,500,000 / [(240 + 5 + 1) \* (320 + 20 + 10)] = **75.5Hz**

# LCD Signals

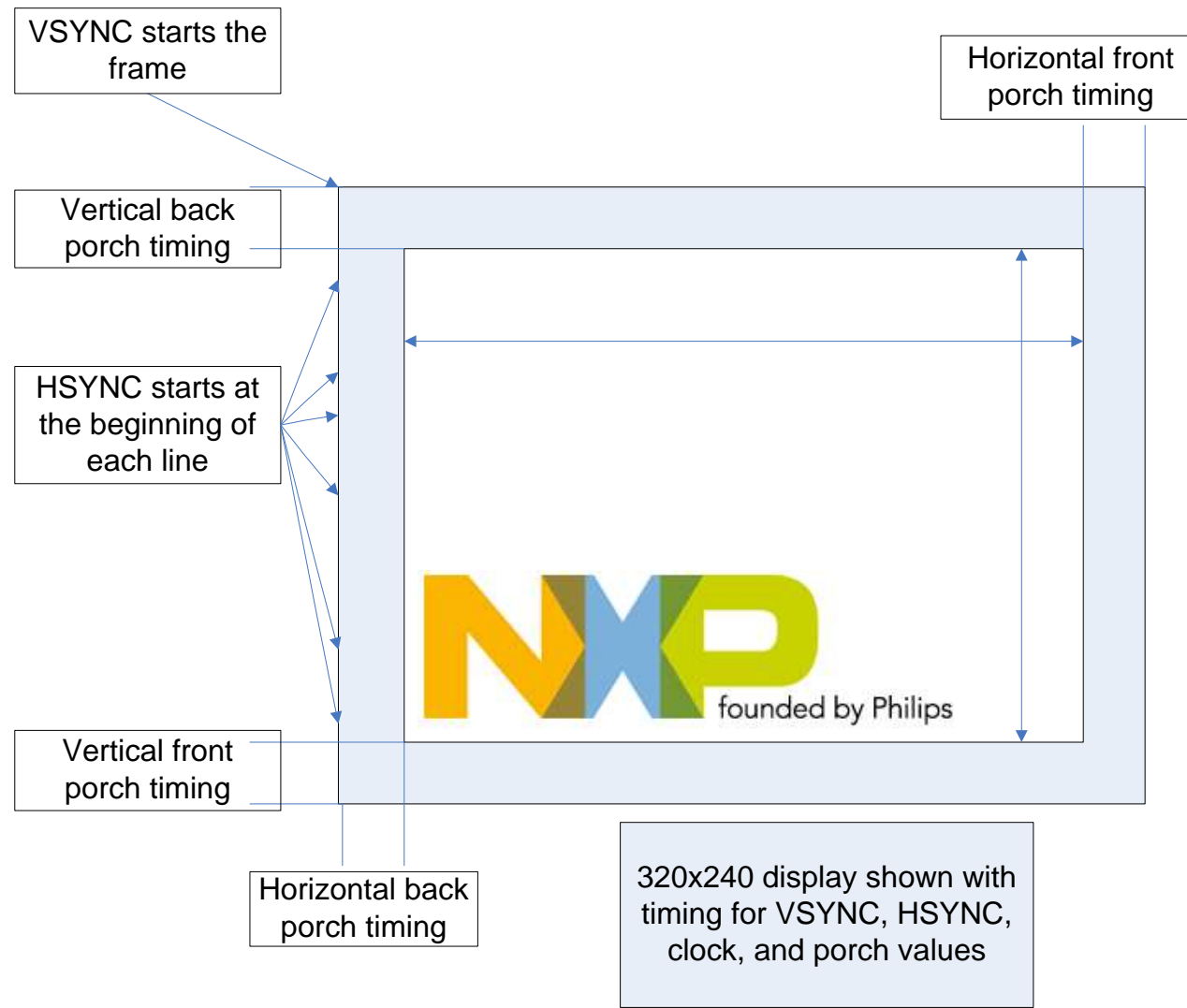
- The largest configuration for the LCD controller uses 31 pins. There are many variants using as few as 10 pins for a monochrome STN panel.

Pin name	Type	Function
LCD_PWR	output	LCD panel power enable.
LCD_DCLK	output	LCD panel clock.
LCD_ENAB_M	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.

# 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_ENAB_M	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

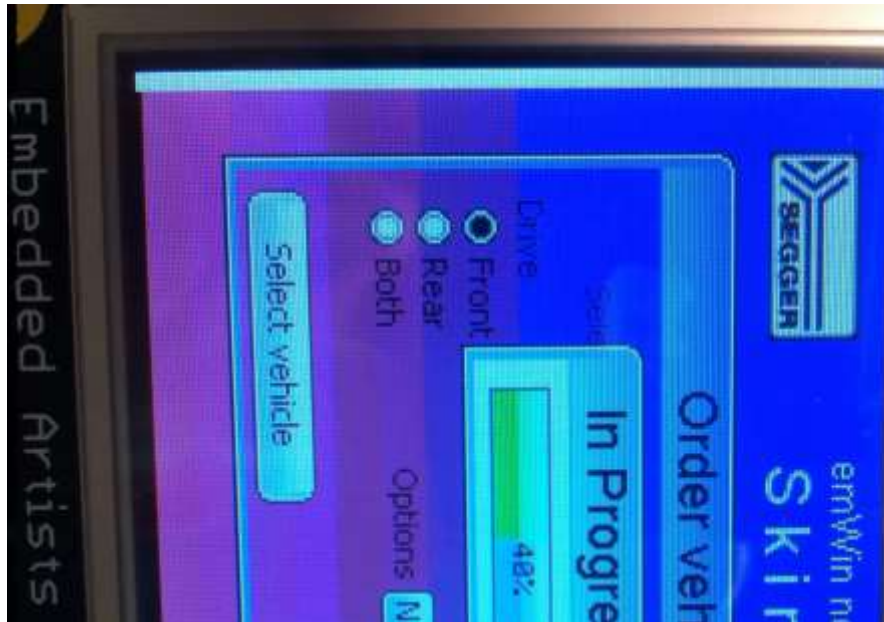




# Example: Truly 240 x 320 TFT RGB666

Characteristics	Symbol	Min	Typ	Max	Unit
DOTCLK Frequency	$f_{\text{DOTCLK}}$	-	5.5	8.22	MHz
DOTCLK Period	$t_{\text{DOTCLK}}$	122	182	-	nSec
Horizontal Frequency (Line)	$f_H$	-	19.6	29.3	kHz
Vertical Frequency (Refresh)	$f_V$	-	60	90	Hz
Horizontal Back Porch	$t_{\text{HBP}}$	-	30	-	$t_{\text{DOTCLK}}$
Horizontal Front Porch	$t_{\text{HFP}}$	-	10	-	$t_{\text{DOTCLK}}$
Horizontal Data Start Point	$t_{\text{HSP}}$	-	30	-	$t_{\text{DOTCLK}}$
Horizontal Blanking Period	$t_{\text{HBP}} + t_{\text{HFP}}$	-	40	-	$t_{\text{DOTCLK}}$
Horizontal Display Area	H <sub>DISP</sub>	-	240	-	$t_{\text{DOTCLK}}$
Horizontal Cycle	$H_{\text{cycle}}$	-	280	-	$t_{\text{DOTCLK}}$
Vertical Back Porch	$t_{\text{VBP}}$	-	4	-	Line
Vertical Front Porch	$t_{\text{VFP}}$	-	2	-	Line
Vertical Data Start Point	$t_{\text{VSP}}$	-	4	-	Line
Vertical Blanking Period	$t_{\text{VBP}} + t_{\text{VFP}}$	-	6	-	Line
Vertical Display Area	V <sub>DISP</sub>	-	320	-	Line
Vertical Cycle	$V_{\text{cycle}}$	-	326	-	Line

# Snapshot of incorrect LCD settings



```
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 <sub>VBP</sub>	-	4
---------------------	------------------	---	---

# 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

# Bus Bandwidth Calculator

LPC178x Bus Bandwidth on Various LCD Resolutions and Color Depths at Various Refresh Rate			
Bus Clock (MHz):		80	
Static External Memory Configuration -			
Bus Width:		32	
Read Delay, WAITRD:		1	
Dynamic External Memory Configuration -			
Bus Width:		32	
Precharge Command Period, t <sub>RP</sub> :		2	
RAS Latency (Active to Read/Write Delay), RAS (t <sub>RCD</sub> ):		2	
CAS Latency, CAS:		2	
LCD Resolution -			
Horizontal (Pixels):		640	
Vertical (Pixels):		480	
Refresh Rate -			
Refresh Rate (Hz):		60	
LCD Color Depths -			
Color Depth (bpp):		16	
Frame Buffer -			
Frame Buffer (KB):		600	
LCD Data Rate -			
Data Rate (Mpixels/s):		18.432	
Data Rate (MWords/s):		9.216	
Data Rate (Mbursts/s):		2.304	
Static External Memory Burst -			
Burst (clocks):		13	
Dynamic External Memory Burst -			
Burst (clocks):		15	
Bus Bandwidth Needed by LCD:			
Static External Memory (%):		37.44	
Dynamic External Memory (%):		43.2	

- <http://www.lpcware.com/content/nxpfile/lcd-bus-bandwidth-calculator-lpc177x8x>



# LPC4088 LCD AHB Priority

- AHB Matrix Arbitration register (Matrix\_Arb - 0x400F C188)
- 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 be lower than PRI_DCODE for proper operation.	0x1
3:2	PRI_DCODE	D-Code bus priority.	0x3
5:4	PRI_SYS	System bus priority.	0
7:6	PRI_GPDMA	General Purpose DMA controller priority.	0
9:8	PRI_ETH	Ethernet DMA priority.	0
11:10	PRI_LCD	LCD DMA priority.	0
13:12	PRI_USB	USB DMA priority.	0
15:14	-	Reserved. Read value is undefined, only zero should be written.	NA
16	ROM_LAT	ROM latency select. Should always be 0.	0
31:17	-	Reserved. Read value is undefined, only zero should be written.	NA

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

# EMWIN FEATURES



# 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.

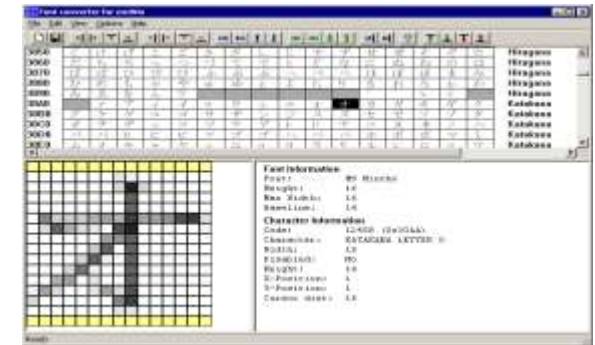
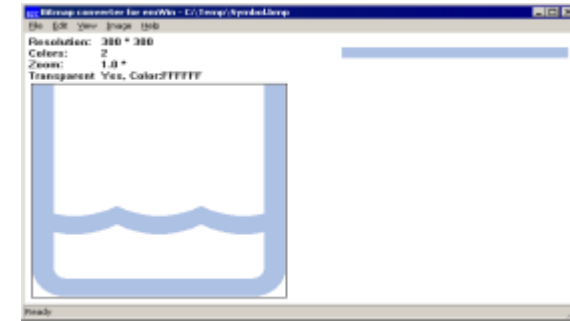
**Bin2C** - Can be used for converting any kind of file into byte arrays which then can be used within a C file.

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

Optional tools are:

**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.



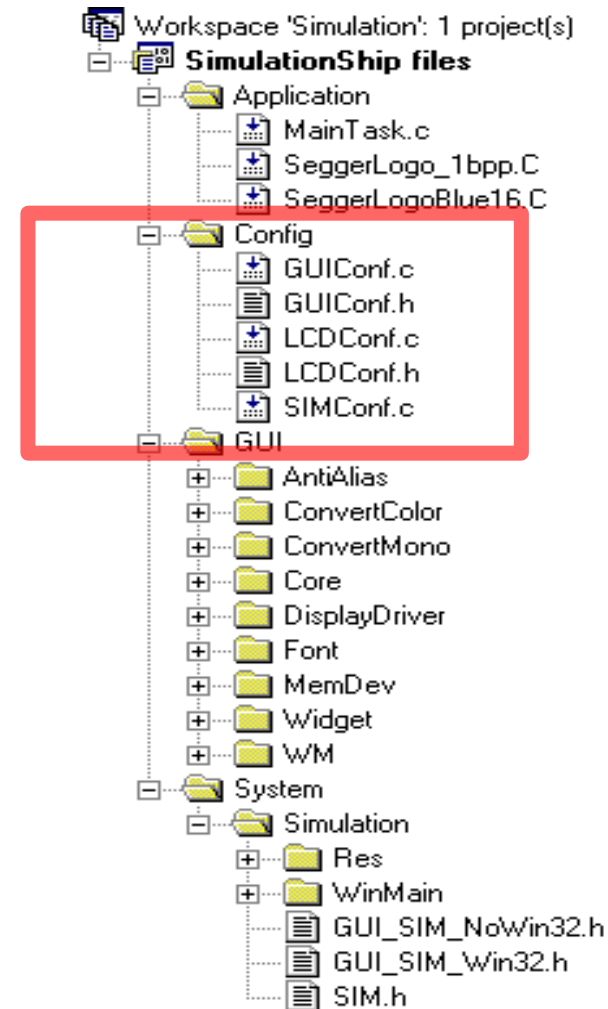
# Configuration

## Compile time configuration (.h files):

- **GUIConf.h** - Configuration of available features
- **LCDConf.h** - Display driver configuration (obsolete)

## Runtime configuration (.c files):

- **GUIConf.c** - Configuration of dynamic memory
- **LCDConf.c** - Display driver configuration and initialization
- **SIMConf.c** - Configuration of simulator
- **GUI\_X.c / GUI\_X\_embOS.c** - Not required in simulation

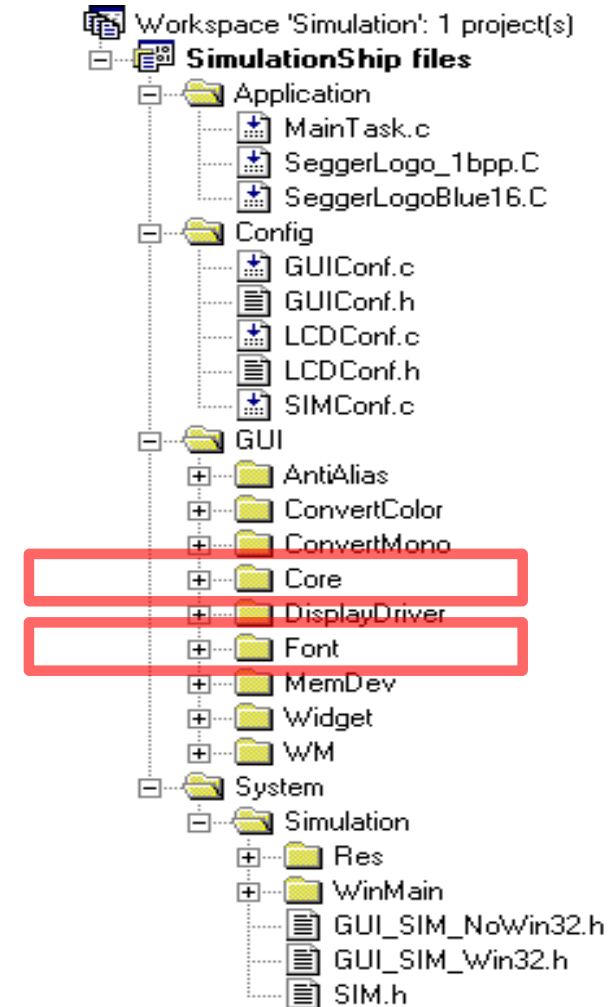




# Core functions

The following features are covered by the base package:

- Image file support for BMP, GIF, PNG and JPEG
- Drawing of images from non addressable media
- LTR, RTL and bidirectional text support
- Software alpha blending
- Sprites and cursors, also animated
- Lines, polygons, rectangles, arcs and circles
- Support for all non antialiased font formats
- Drawing of values (dec, bin, hex and float)
- Multiple buffering
- Virtual screens
- Touch screen support
- Multitasking support
- Standard font package (ASCII and ISO 8859-1)



# Memory devices

## How do they work?

- Drawing operations can be passed to a memory device instead to the display. 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 means 'transparency' 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.

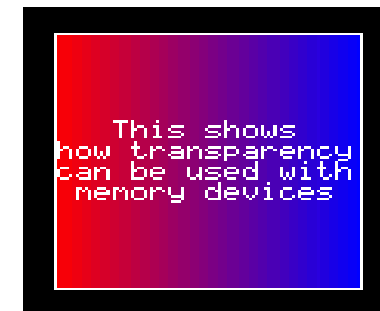
## Do memory devices support alpha blending?

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);
    }
}
```



# Antialiasing

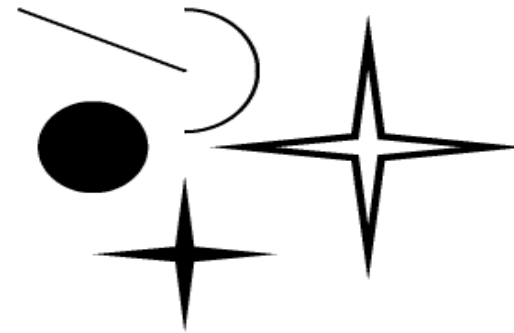
- Antialiasing smoothes curves and diagonal lines by "blending" the background color with that of the foreground.
- emWin supports antialiased drawing of
  - **Text**  
Font converter is required for creating AA fonts.
  - **Arcs**  
`GUI_AA_DrawArc()`
  - **Circles**  
`GUI_AA_FillCircle()`
  - **Lines**  
`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

## What is the **Window Manager**?

- **Management system for a hierarchic window structure**  
Each layer has its own desktop window. Each desktop window can have its own hierarchic 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, 319, 239, 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);
    }
}
```



# 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:

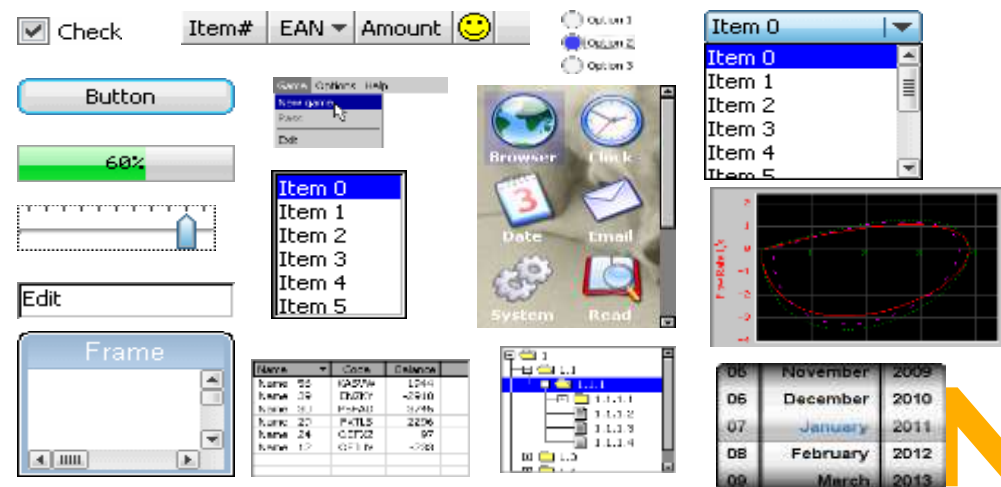
- `<WIDGET>_CreateIndirect()`  
Creation by dialog box creation function.

## Direct creation

```
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

```
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, 0, 0, 0);  
    while (1) {  
        GUI_Delay(100);  
    }  
}
```



# 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()
```

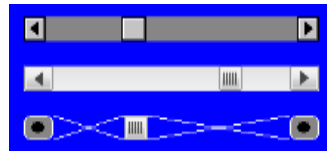
The skin of each single widget can be set by:

```
<WIDGET>_SetSkin()
```

Properties of FLEX\_SKIN can be fetched / set by:

```
<WIDGET>_GetSkinFlexProps()
```

```
<WIDGET>_SetSkinFlexProps()
```



```
#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;
}
```

SAMPLE: Skinning\MainTask\_ScrollBarSkin.c



# GUI-Builder

The GUI-Builder is a tool for creating dialogs without any knowledge of the C programming language.

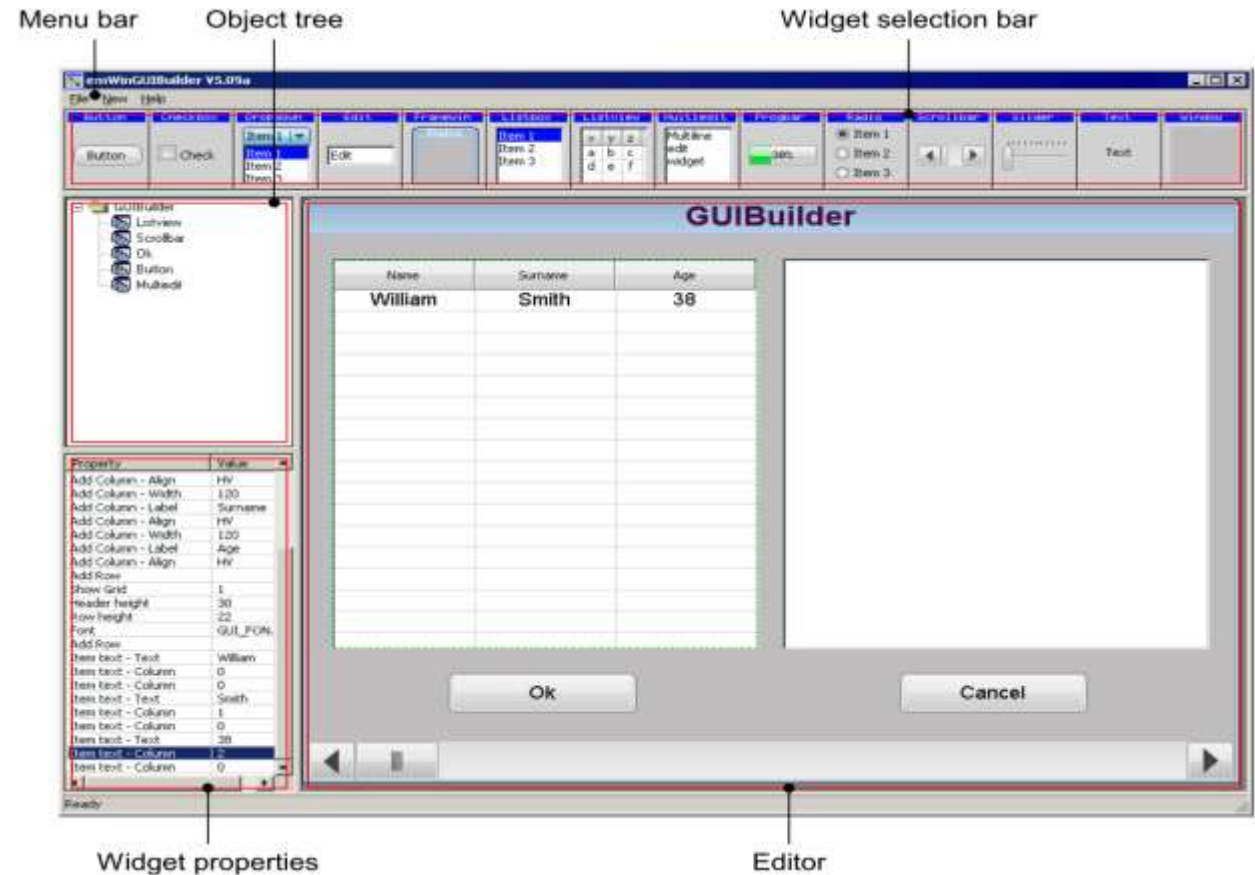
Basic usage of the GUI-Builder:

- **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`

Property	Value
Name	Framewin
xPos	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**



# Common dialogs

Common dialogs are available for

- Message boxes
- Color selection
- Exploring a file system

## MESSAGEBOX

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

## CHOOSEFILE

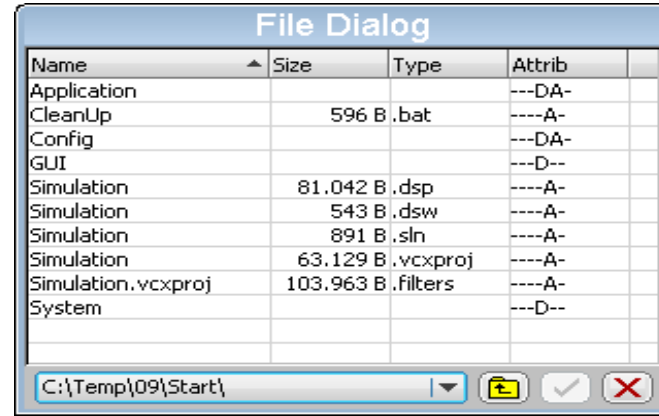
- Embedded file system explorer.
- Simple callback mechanism used to get file data from application.
- Ready to use sample for emFile available

## CHOOSECOLOR

- Selection from an application defined array of colors.

## CALENDAR

- Selection of a date of the gregorian calendar.





# What is Available?

- Board Support Packages (BSPs) available for:
  - Embedded Artists LPC4088 Display Module
  - Keil MCB1700 – using LPC1769
  - Embedded Artists LPC1788 – using LPC1788
  - IAR 1788-SK – using LPC1788
  - Supported compilers: Keil  $\mu$ Vision / IAR EWARM / LPCXpresso / MS Visual C++
- Libraries for ARM926/ARM7/Cortex-M0/M3/M4
- Manuals/Guides:
  - Start-up guide – how to use emWin on NXP microcontrollers
  - Porting guide – how to port a BSP to new hardware/LCD's
  - emWin manual – over 1000 pages on all emWin features
- <http://www.lpcware.com/content/project/emwin-graphics-library>



SECURE CONNECTIONS  
FOR A SMARTER WORLD