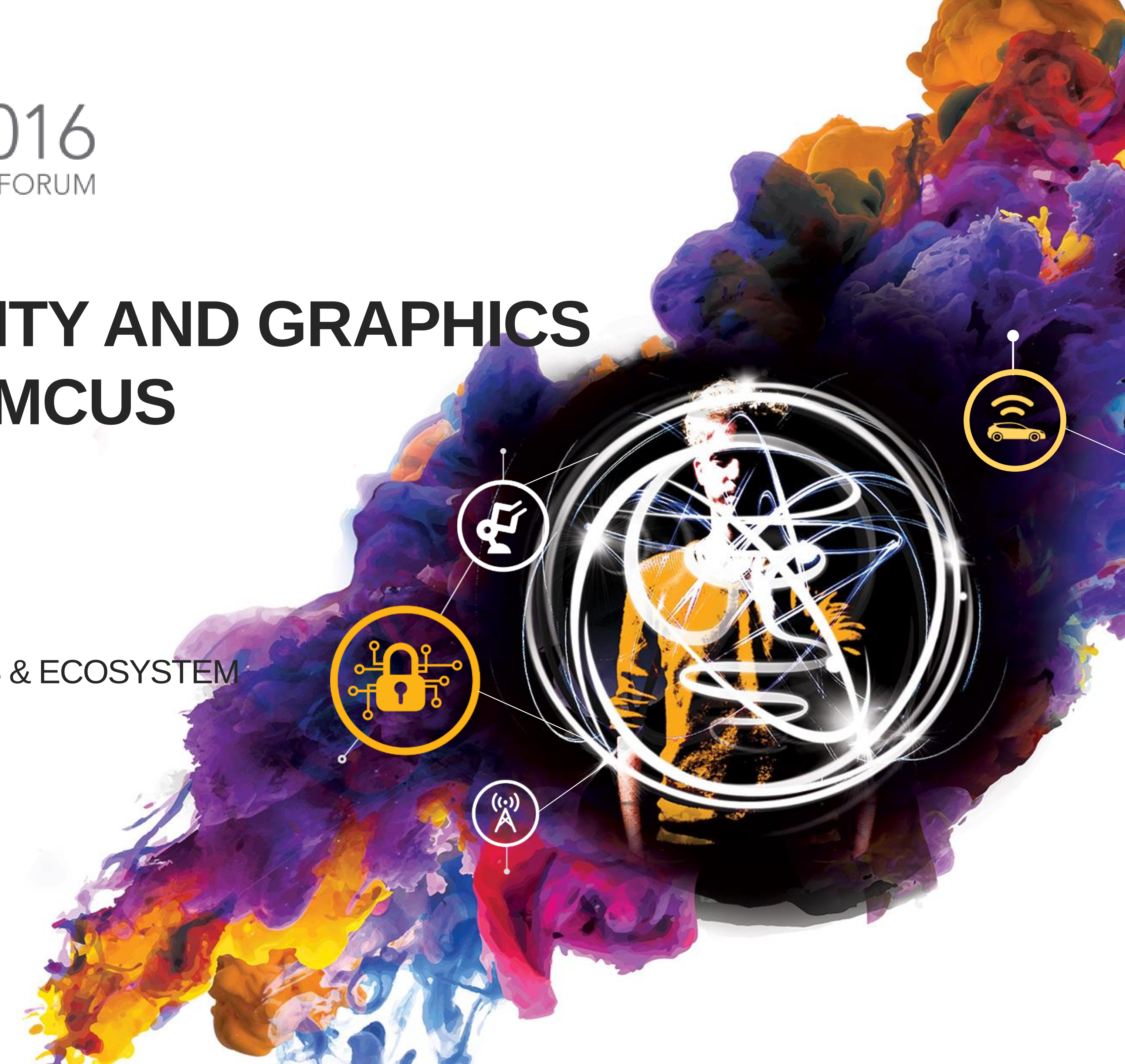
NXP | FTF 2016 TECHNOLOGY FORUM

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 USE

AGENDA

- Session Objectives
- Development Platform Overview
- LPC4088 Device and Embedded Artists Display Module
- LPC Ecosystem and the LPCXpresso IDE
  - Loading, Building and Running Your First LPCOpen Example
- Graphics Solutions
  - Application of Graphic Displays
  - Choosing a Graphics Package
  - emWIN Technical Session, including USB connectivity

SESSION OBJECTIVES

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

NXP LPC Microcontroller Portfolio At-a-Glance

From entry level

Easy to use
Exceptional power efficiency
Lowest pin count

To high performance

Best power efficiency
Advanced connectivity
Flexible peripherals

LPC800 Series

Low power, basic control and connectivity

- 30 MHz Cortex-M0+ core
- Basic serial connectivity
- Basic analog
- Low-pincount packages including TSSOP and HVQFN and XSON
- Ideal for 8-/16-bit transition

LEARN MORE

LPC1100 Series

Power efficient, broad selection, industry-standard connectivity

- 50 MHz Cortex-M0+ & M0 cores
- Serial connectivity: USB with PHY, CAN with transceiver
- Best-in-class analog
- Broad package selection
- Migration path to LPC1300 Series

LEARN MORE

LPC1200 Series

Noise immunity for industrial applications

- 45 MHz Cortex-M0 core
- High-immunity rating (IEC61697-1)
- 8 kV ESD protection
- Basic analog
- Real-time clock
- Fm I²C with 10x bus-drive capability

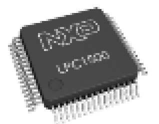
LEARN MORE

LPC1300 Series

Performance and basic connectivity

- Up to 72 MHz Cortex-M3 core
- Serial connectivity: USB, CAN
- Pin-compatible upgrade for most LPC1100 Series devices

LEARN MORE

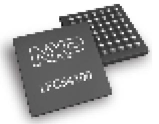


LPC1500 Series

High-precision motion control

- Up to 72 MHz Cortex-M3 core
- Optimized for sensored & sensorless brushless motor control; free FOC firmware
- Serial connectivity: USB, CAN
- Advanced analog subsystem and SCTimer/PWM

LEARN MORE



LPC54100 Series

Ultra-low-power for always-on sensor processing

- Up to 100 MHz single- & dual-core: Cortex-M4F & M0+ (opt.)
- Optimized for sensor listening, aggregation, fusion, and communication
- Ultra-low 'power down' mode, down to 3 µA for sensor listening
- Scalable power performance

LEARN MORE

LPC1700 Series

LPC4000 Series

High performance with DSP options, multi-connectivity, advanced peripherals

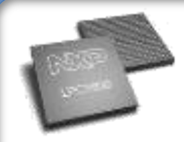
LPC1700 Series:

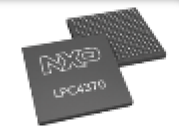
- Up to 120 MHz Cortex-M3 core
- Advanced connectivity: USB, CAN, Ethernet
- Graphic LCD controller
- Pin-compatible migration path to LPC4000 Series and ARM7 LPC2x00 Series

LPC4000 Series:

- Up to 120 MHz Cortex-M4/M4F cores with DSP
- Advanced conn.: USB, CAN, Ethernet
- Graphic LCD controller
- Analog comparators
- Drop-in perf. upgrade for LPC1700 and LPC2x00 series

LEARN MORE





LPC1800 Series / LPC18Sxx Family

LPC4300 Series / LPC43Sxx Family

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

LPC1800 Series:

- 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

LPC4300 Series:

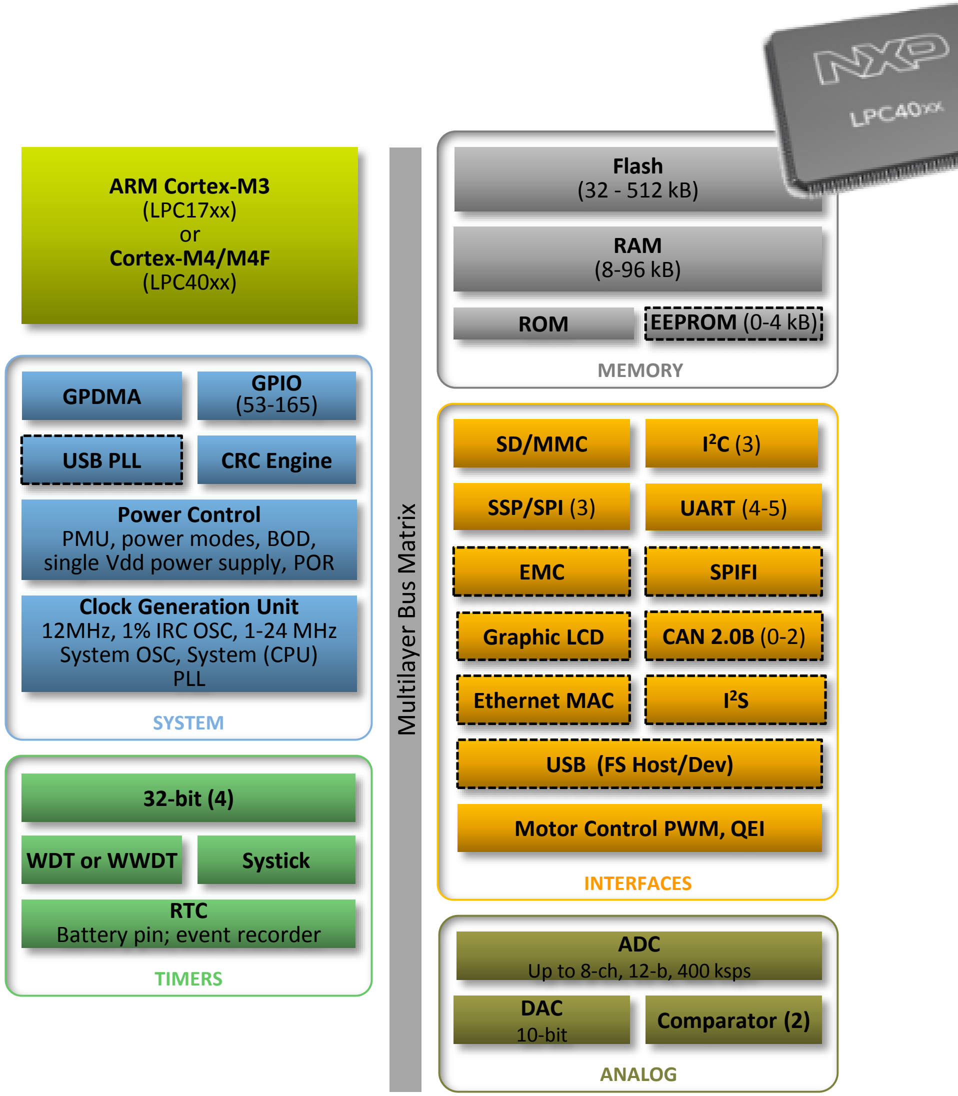
- Up to 204 MHz Cortex-M4F core with DSP capabilities and Cortex-M0 coprocessor(s)
- LPC43Sxx family includes integrated security features
- Partition tasks across cores to optimize performance
- Advanced conn.: dual Hi-Speed USB, dual CAN, 10/100 Ethernet, configurable high-speed serial I/O
- Best-in-class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

LEARN MORE

LPC1700/4000 Series

High performance, multi-connectivity, advanced peripherals

- 120 MHz Cortex-M3 or Cortex-M4/M4F
  - 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



ARM Cortex-M3 (LPC17xx) or **Cortex-M4/M4F** (LPC40xx)

GPDMA

GPIO (53-165)

USB PLL

CRC Engine

Power Control
PMU, power modes, BOD, single Vdd power supply, POR

Clock Generation Unit
12MHz, 1% IRC OSC, 1-24 MHz System OSC, System (CPU) PLL

SYSTEM

32-bit (4)

WDT or WWDT

Systick

RTC
Battery pin; event recorder

TIMERS

Multilayer Bus Matrix

Flash (32 - 512 kB)

RAM (8-96 kB)

ROM

EEPROM (0-4 kB)

MEMORY

SD/MMC

I^{2}C (3)

SSP/SPI (3)

UART (4-5)

EMC

SPIFI

Graphic LCD

CAN 2.0B (0-2)

Ethernet MAC

I^{2}S

USB (FS Host/Dev)

Motor Control PWM, QEI

INTERFACES

ADC
Up to 8-ch, 12-b, 400 ksps

DAC
10-bit

Comparator (2)

ANALOG

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

Portfolio Breadth & Scalability

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

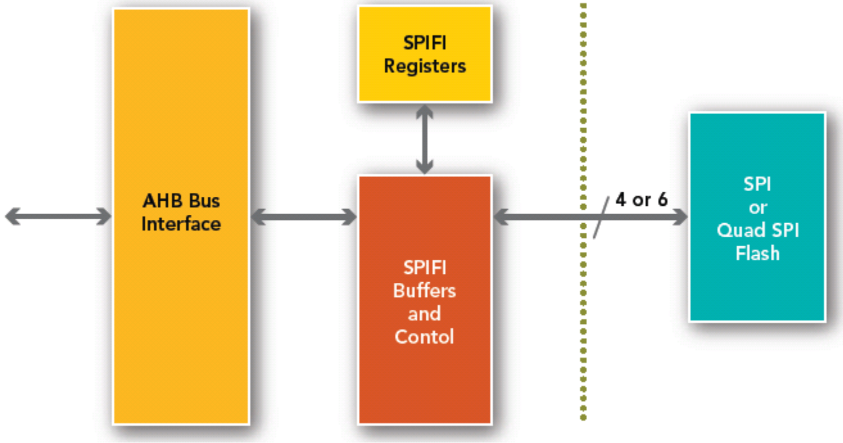
	Mainstream MCU	High Performance MCU	High performance & Large RAM MCU/MPU
Cortex-A7			528MHz iMX6UL
ARM9			270MHz LPC3000
Cortex-M4	180MHz LPC5460x SPIFI 120MHz LPC4000 SPIFI	204MHz LPC4300 SPIFI	LPC4300 SPIFI
Cortex-M3	120MHz LPC1700	180MHz LPC1800 SPIFI	LPC1800 SPIFI
ARM7	72MHz LPC2400		
	512KB Flash	1MB Flash	Flash-less

Connectivity Peripherals – USB

- Features
  - USB 2.0 host/device/OTG
  - USB Low-Speed, Full-Speed and Hi-Speed USB w/ integrated PHY
  - All endpoint types (control, bulk, interrupt, isochronous)
  - 2nd PLL used for USB – core & USB can run at different clock
  - OHCI/EHCI-compliant host controller
  - All USB parts USB-IF certified
  - Integrated DMA support
  - USB drivers in ROM
- NXP VID/PID program

Memory

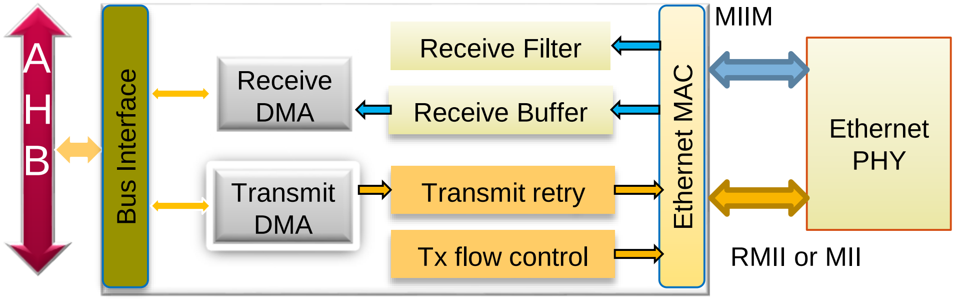
SPIFI (SPI Flash Interface)



- 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

Connectivity Peripherals – Ethernet

- 10/100 Mbps IEEE 802.3 Ethernet MAC
- IEEE 1588-2008 time stamping block
- Supports both full-duplex and half-duplex operation
- DMA support, dedicated packet RAM maximizes performance
- External MII and RMII Ethernet PHY
- LWIP stack supported in the LPCOpen software platform
- Better performance through independent transmit and receive buffers



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

Walk around the board

LPC ECOSYSTEM AND THE LPCXPRESSO IDE

LPC Microcontroller Ecosystem

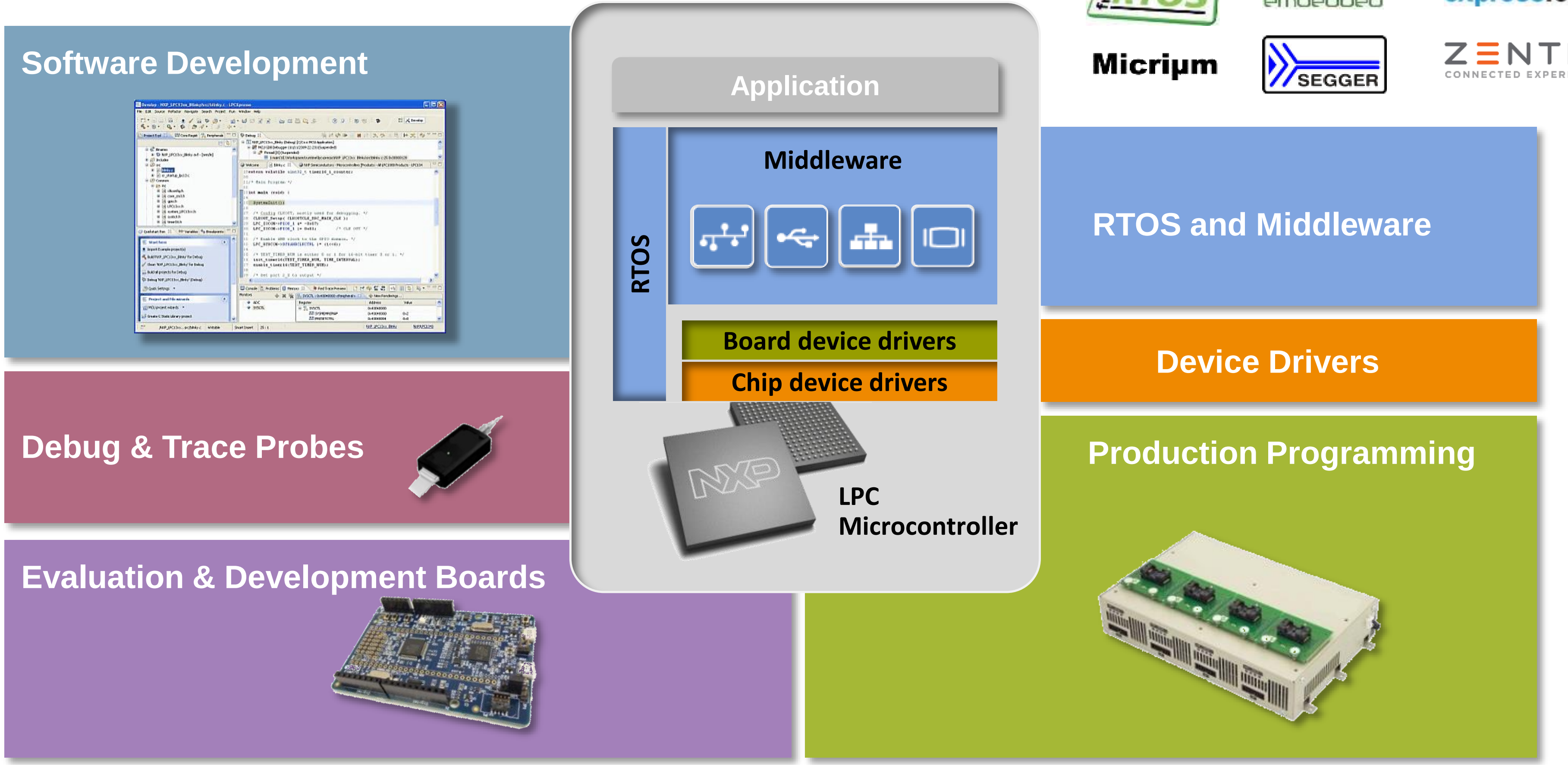
KEIL Tools by ARM

Rowley Associates

IAR SYSTEMS

LAUTERBACH

i SYSTEM



Software Development

Debug & Trace Probes

Evaluation & Development Boards

Application

RTOS

Middleware

Board device drivers

Chip device drivers

LPC Microcontroller

RTOS and Middleware

Device Drivers

Production Programming

freeRTOS

mentor embedded

expresslogic

altia

Micrium

SEGGER

ZENTRI CONNECTED EXPERIENCE

TouchGFX

wolfSSL

interniche technologies, inc.

Cypherbridge Systems

HCC embedded

KEIL Tools by ARM

Embedded Artists

FDI

hitex DEVELOPMENT TOOLS

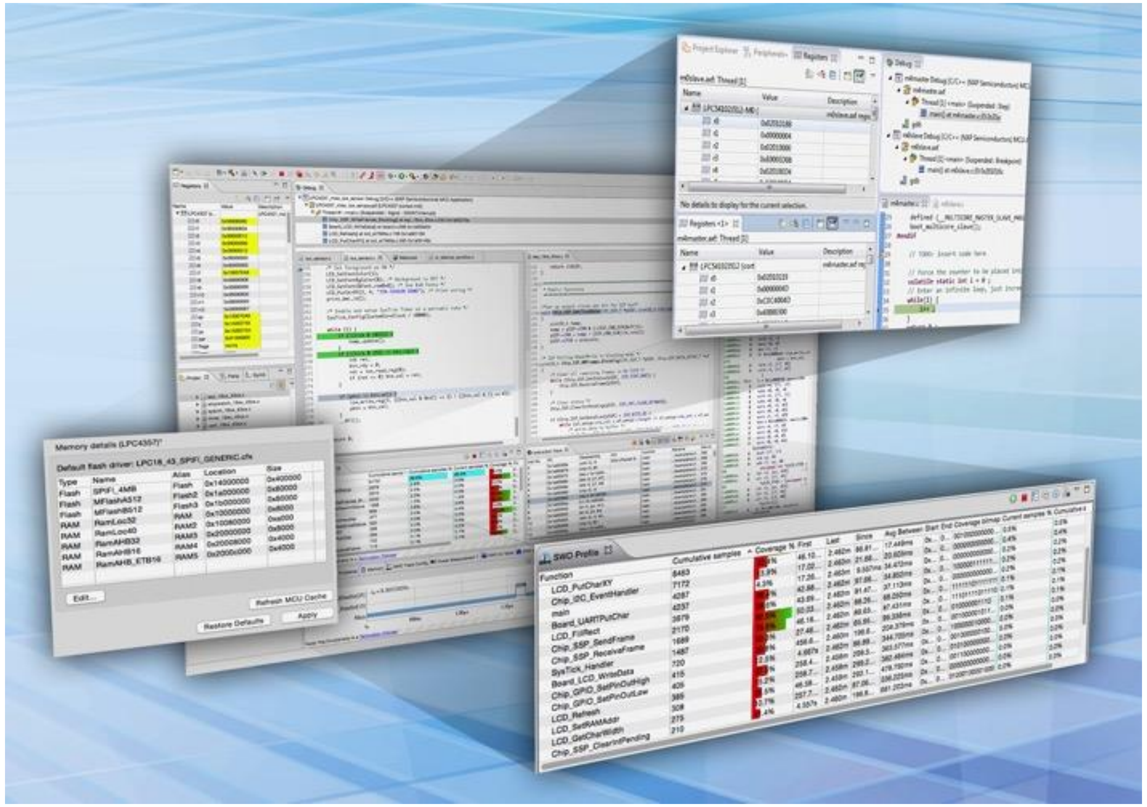
NGX TECHNOLOGIES

What is “LPCXpresso”?

- The LPCXpresso brand is used for NXP’s in-house, Integrated Development Environment and low cost , highly flexible development *boards* for LPC MCUs
- Closely related, but independent:
  - LPCXpresso *IDE* can be used with LPCXpresso boards or any other LPC target system
  - LPCXpresso *boards* can be used with LPCXpresso IDE or other development tools

LPCXpresso Integrated Development Environment (IDE)

- Enhanced Eclipse / GCC based IDE
  - Focused on ease of use for LPC MCUs
- Cross Platform:
  - Windows, Mac OS X and Linux versions
- LPC-Link/LPC-Link2 and Segger J-link probe support
- Free edition
  - 256KB download limit
  - forum support
  - Simple registration online to enable
- Pro edition ($495)
  - Unlimited code size
  - Professional support and enhanced trace capability
  - Enhanced trace features
  - Upgrade from Free version with simple license update



Memory details (LPC4357)

Default flash driver: LPC18_43_SPIFI_GENERIC.cfx

Type	Name	Alias	Location	Size
Flash	SPIFI_4MB	Flash	0x14000000	0x400000
Flash	MFlashA512	Flash2	0x1a000000	0x80000
Flash	MFlashB512	Flash3	0x1b000000	0x80000
RAM	RamLoc32	RAM	0x10000000	0x8000
RAM	RamLoc40	RAM2	0x10080000	0xa000
RAM	RamAHB32	RAM3	0x20000000	0x8000
RAM	RamAHB16	RAM4	0x20008000	0x4000
RAM	RamAHB_ETB16	RAM5	0x2000c000	0x4000

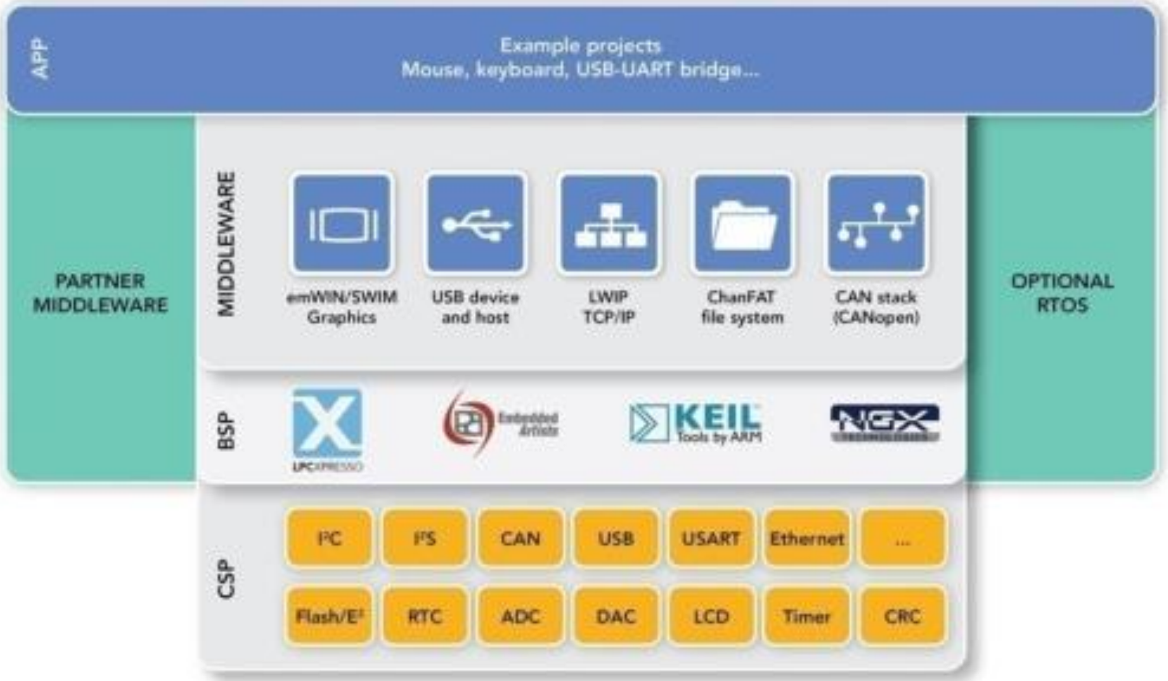
SWO Profile

Function	Cumulative samples
LCD_PutCharXY	8463
Chip_I2C_EventHandler	7172
main	4287
Board_UARTPutChar	4237
LCD_FillRect	3879
Chip_SSP_SendFrame	2170
Chip_SSP_ReceiveFrame	1689
SysTick_Handler	1487
Board_LCD_WriteData	720
Chip_GPIO_SetPinOutHigh	415
Chip_GPIO_SetPinOutLow	406
LCD_Refresh	385
LCD_SetRAMAddr	308
LCD_GetCharWidth	275
Chip_SSP_ClearIntPending	210

- Available for free download at www.nxp.com/lpcxpressoide

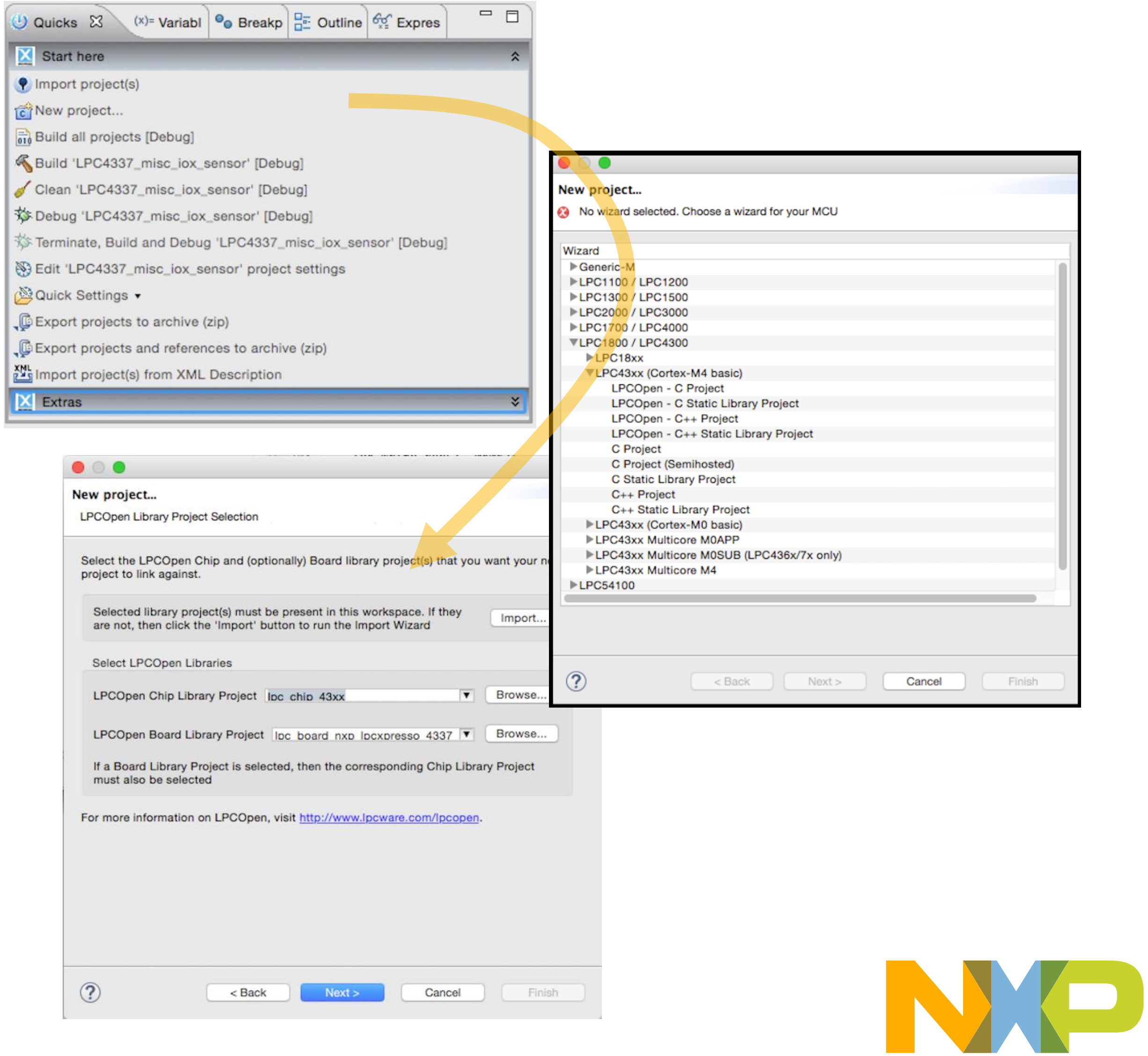
LPCOpen – free drivers and examples

- Comprehensive set of RTOS-agnostic LPC libraries
  - Create multi-functional products
  - Chip and board support packages
  - Includes examples for FreeRTOS, usable with any RTOS
  - Uses common APIs and can be built with Keil, IAR, and LPCXpresso tool chains
- Combines device drivers with stacks
  - SEGGER emWin and SWIM graphics libraries
  - LWIP IP stack
  - USB slave and host
  - CANOpen
- End-application ready
  - Architected for real system use
  - Built & tested with full optimization
  - Meaningful examples



LPCXpresso Ease of use – Quick start panel and project Wizards

- LPCXpresso IDE enhances Eclipse features to simplify user experience
  - Quick start panel provides easy, one-click access to common operations
  - Project wizards provide simple import of LPCOpen libraries, and easy creation of projects using (optionally) these drivers



Start here

- Import project(s)
- New project...
- Build all projects [Debug]
- Build 'LPC4337_misc_iox_sensor' [Debug]
- Clean 'LPC4337_misc_iox_sensor' [Debug]
- Debug 'LPC4337_misc_iox_sensor' [Debug]
- Terminate, Build and Debug 'LPC4337_misc_iox_sensor' [Debug]
- Edit 'LPC4337_misc_iox_sensor' project settings
- Quick Settings ▾
- Export projects to archive (zip)
- Export projects and references to archive (zip)
- Import project(s) from XML Description

Extras

New project...

No wizard selected. Choose a wizard for your MCU

Wizard

- ▶Generic-M
- ▶LPC1100 / LPC1200
- ▶LPC1300 / LPC1500
- ▶LPC2000 / LPC3000
- ▶LPC1700 / LPC4000
- ▼LPC1800 / LPC4300
  - ▶LPC18xx
  - ▼LPC43xx (Cortex-M4 basic)
    - LPCOpen - C Project
    - LPCOpen - C Static Library Project
    - LPCOpen - C++ Project
    - LPCOpen - C++ Static Library Project
    - C Project
    - C Project (Semihosted)
    - C Static Library Project
    - C++ Project
    - C++ Static Library Project
  - ▶LPC43xx (Cortex-M0 basic)
  - ▶LPC43xx Multicore M0APP
  - ▶LPC43xx Multicore M0SUB (LPC436x/7x only)
  - ▶LPC43xx Multicore M4
- ▶LPC54100

New project...

LPCOpen Library Project Selection

Select the LPCOpen Chip and (optionally) Board library project(s) that you want your n project to link against.

Selected library project(s) must be present in this workspace. If they are not, then click the 'Import' button to run the Import Wizard

Select LPCOpen Libraries

LPCOpen Chip Library Project: lpc chip 43xx

LPCOpen Board Library Project: lpc board nxp lpcxpresso 4337

If a Board Library Project is selected, then the corresponding Chip Library Project must also be selected

For more information on LPCOpen, visit http://www.lpcware.com/lpcopen.

HANDS-ON SESSION 1

LPCXpresso IDE – introductory example

- Hands-on session with following objectives:
  - Get familiar with importing LPCOpen into LPCXpresso IDE
  - Building and running a basic example (blinky) on the LPC4088 Display Module
  - Get familiar with debugging operations

APPLICATION OF DISPLAYS

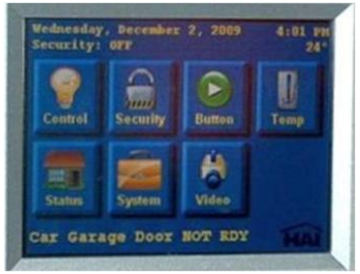
Graphic versus Segment LCD

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

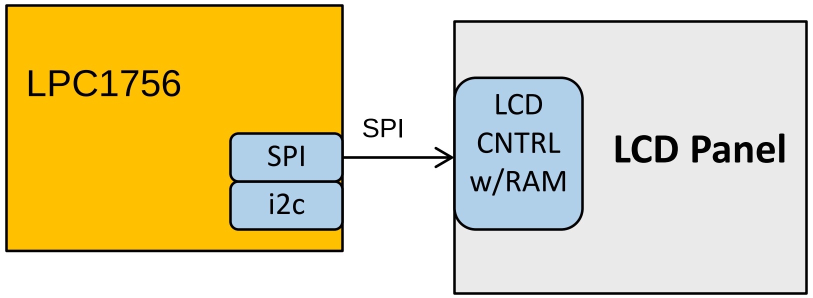


- Embedded applications for vibrant displays.
  - Home Automation and Security
    - Thermostats, security panel, intercom
  - Secure Transactions
    - POS Systems, Access Control, Ticketing
  - White Goods
    - High end Display and Human Interface
  - Industrial Human Machine Interface/ Programmable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

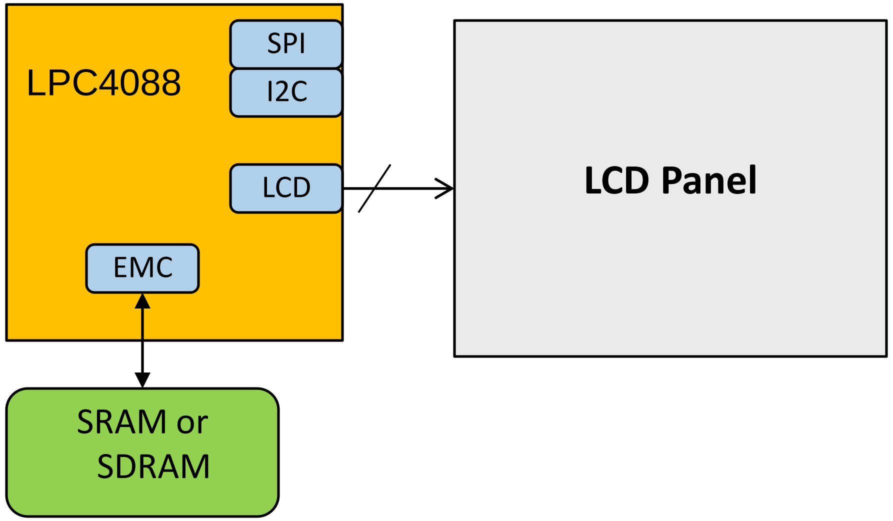
MCU with external or internal LCD controller

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.

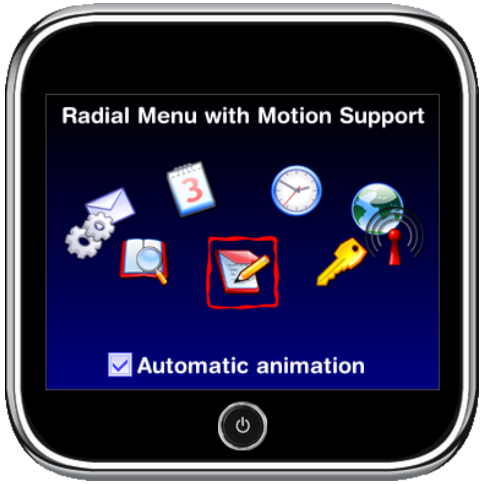


CHOOSING A GRAPHICS PACKAGE

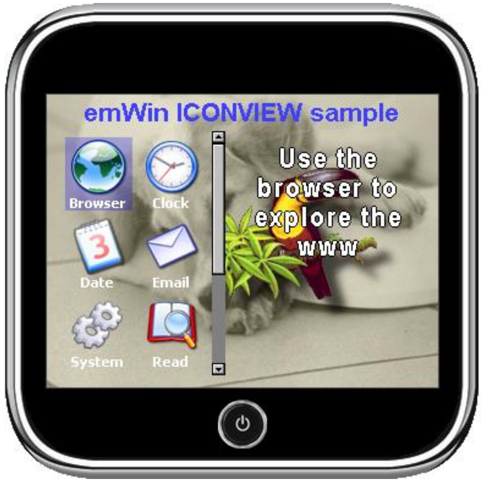
Challenges

- 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

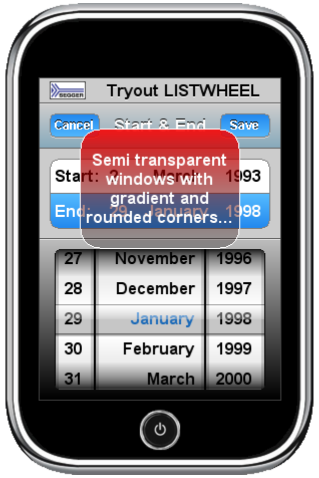
Typical emWin HMI's


Radial Menu with Motion Support

☑ Automatic animation

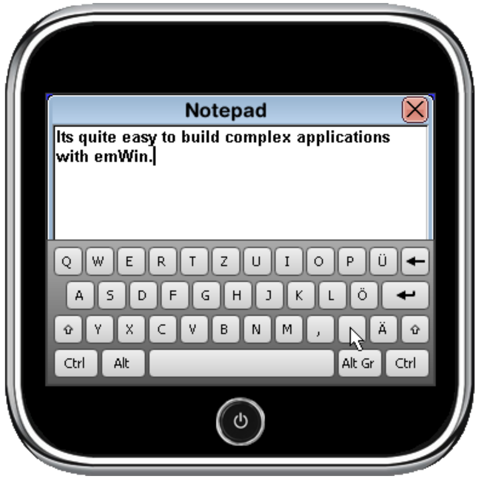

emWin ICONVIEW sample

Use the browser to explore the www

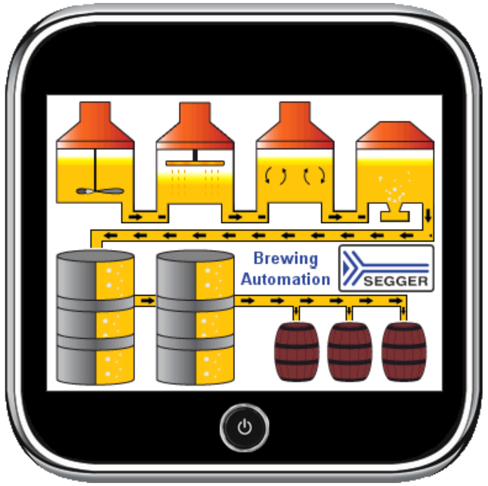

Tryout LISTWHEEL

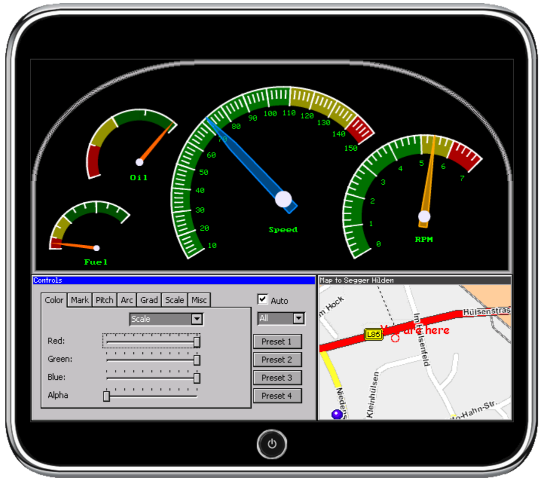
Semi transparent windows with gradient and rounded corners...

27	November	1996
28	December	1997
29	January	1998
30	February	1999
31	March	2000


Notepad

Its quite easy to build complex applications with emWin.





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

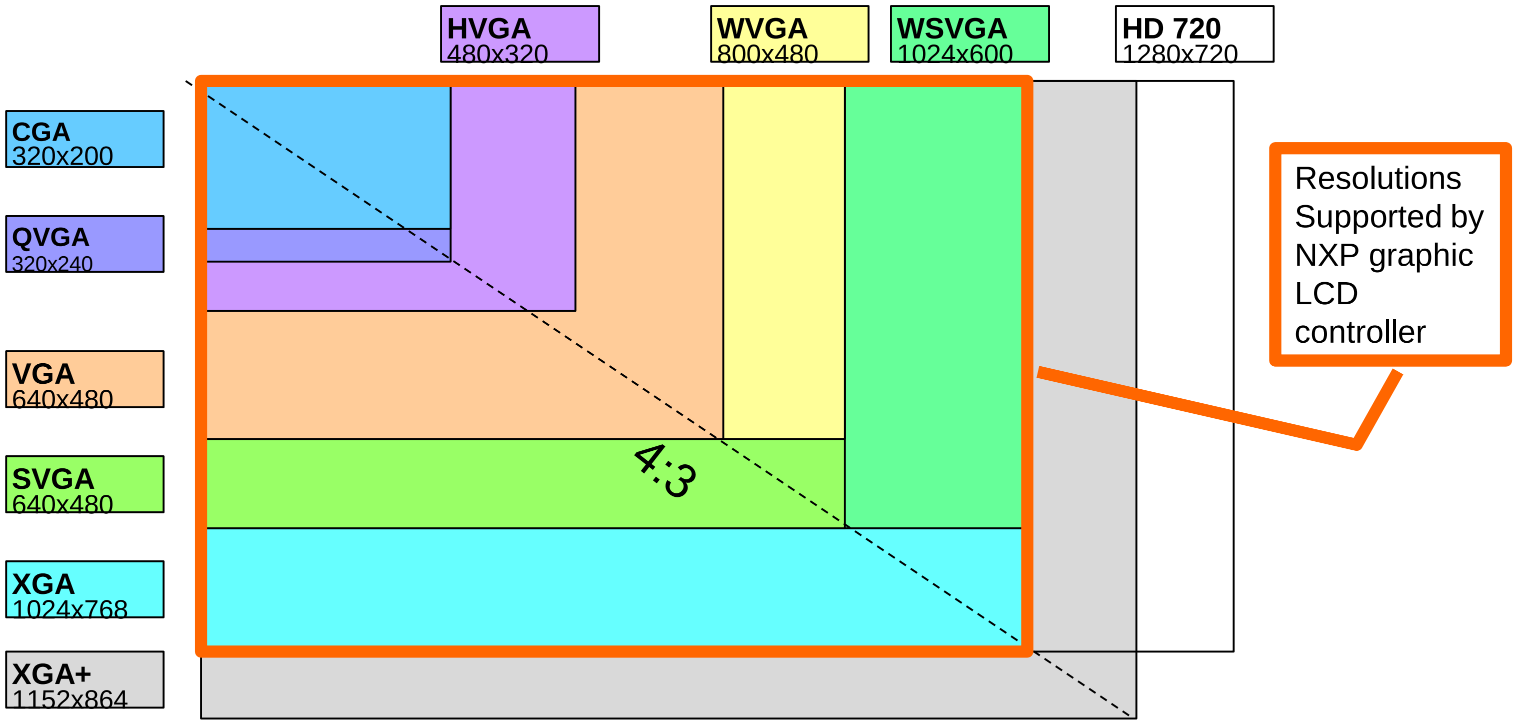
- Free to use with any current NXP ARM Cortex M0, M3 or M4 MCU
- No royalties or licensing fees when used with NXP MCUs
- No source (provided as a pre-compiled library)
- Source code available under license agreement from SEGGER

- 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

28

TECHNICAL SESSION

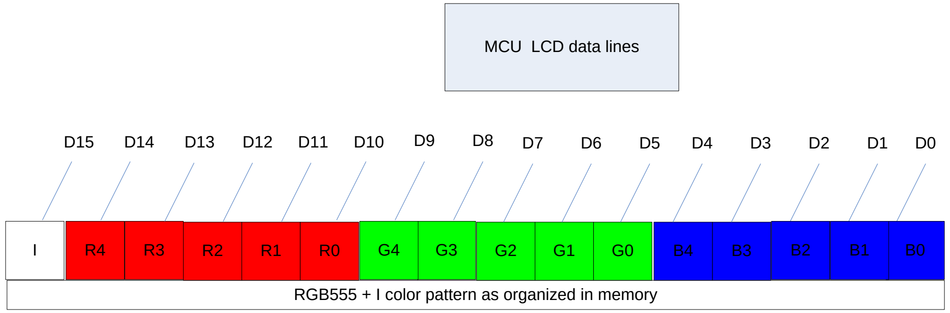
Choosing an LCD: Resolution:



Resolutions Supported by NXP graphic LCD controller

Choosing an LCD: Color Depth

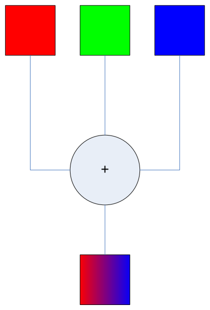
- Color depth or bits per pixel (bpp)



MCU LCD data lines

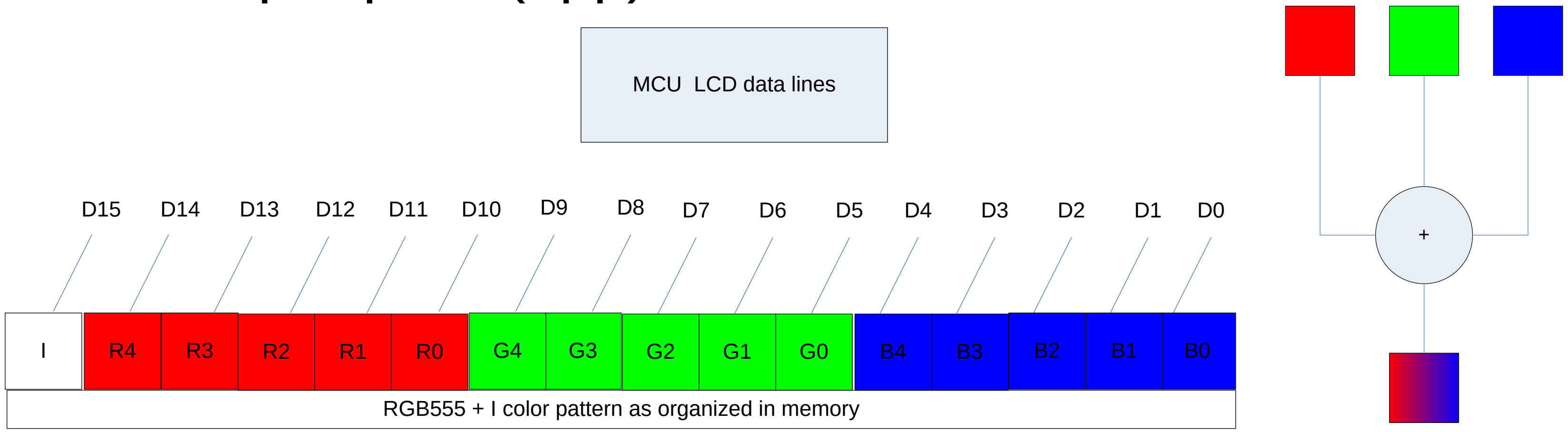
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
I	R4	R3	R2	R1	R0	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0

RGB555 + I color pattern as organized in memory



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)



D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
I	R4	R3	R2	R1	R0	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0

RGB555 + I color pattern as organized in memory

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

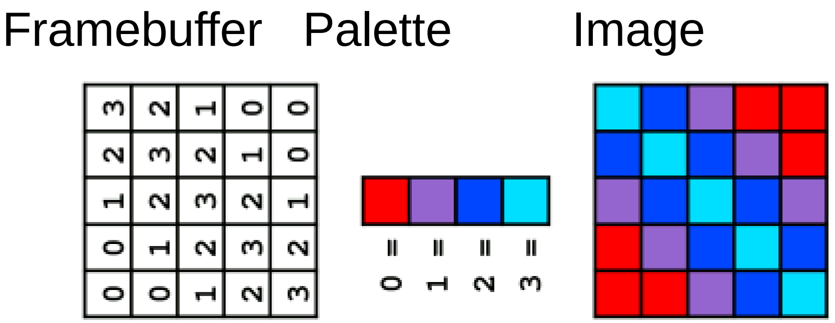
- Resolution x Color Depth = total bits needed (divide x8 for bytes)
- Framebuffer = memory buffer containing complete frame (bitmap) of data

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

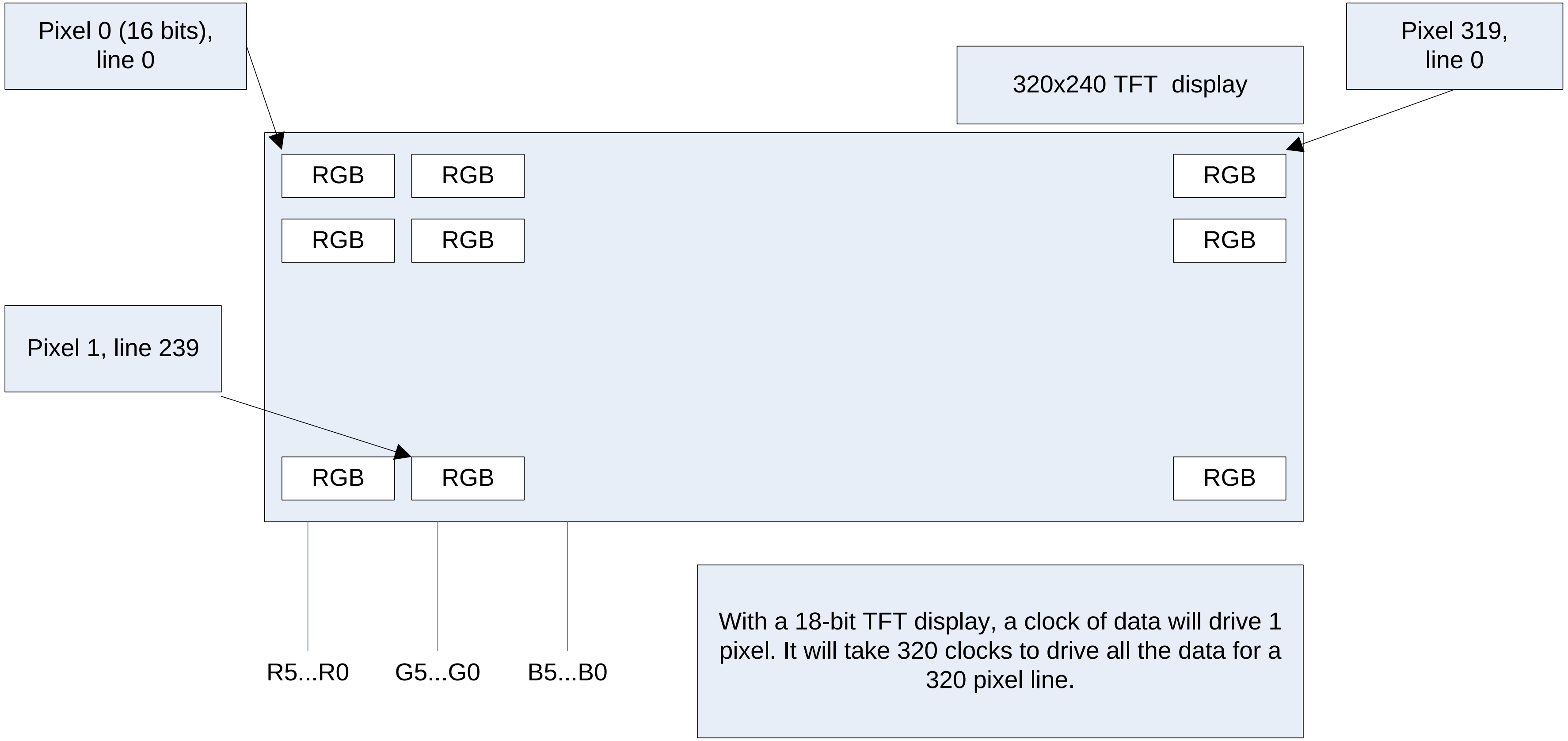
- The frame buffer will contain an index value for each pixel
- Palette RAM is pre-filled with 16-bit color value for each index



3	2	1	0	0
2	3	2	1	0
1	2	3	2	1
0	1	2	3	2
0	0	1	2	3

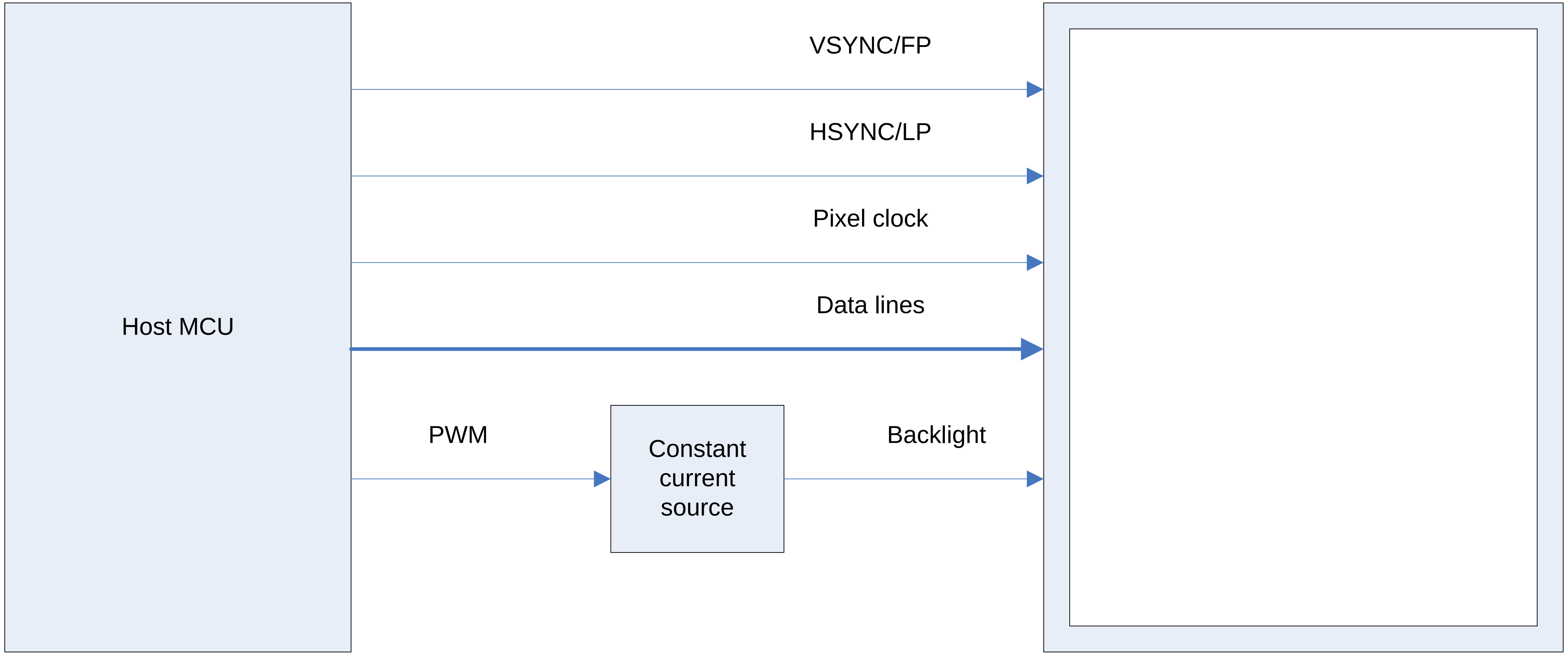
- NXP microcontrollers have 256 entries to support
  - 1, 2, 4, or 8 bpp palletized color displays for color STN and TFT
  - 1, 2, or 4 bits-per-pixel (bpp) palletized displays for mono STN

LCD Clocked TFT



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) =

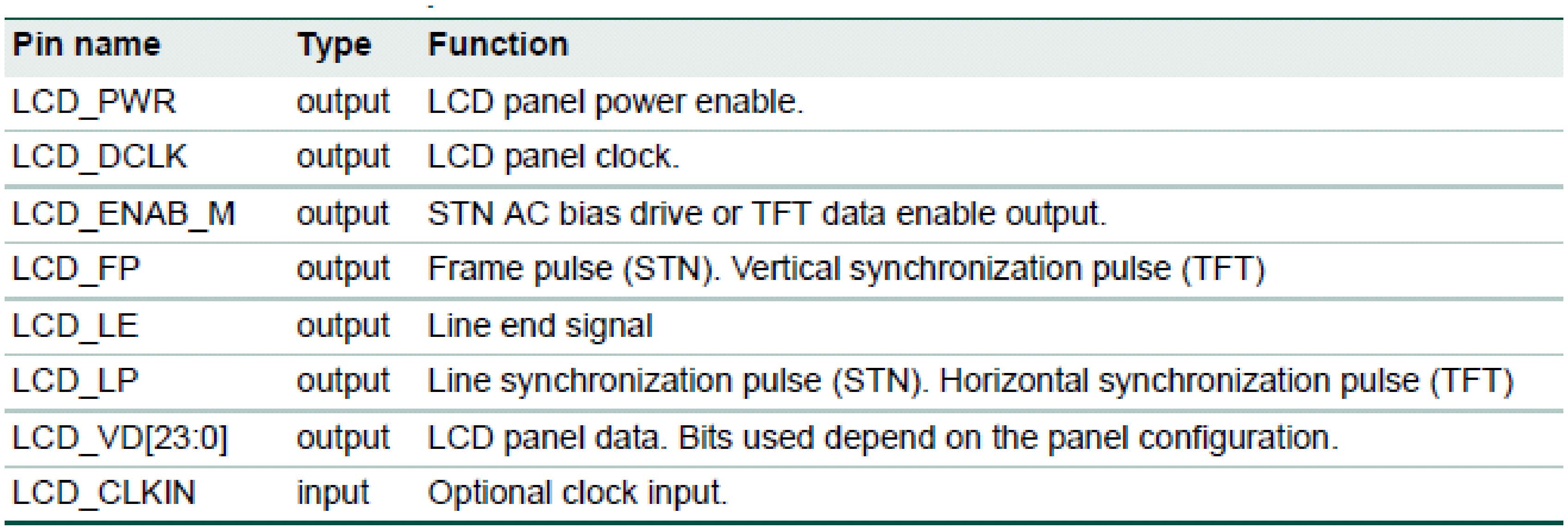
  pixel_clock_rate / [(vertical_resolution + vertical_front_porch + vertical_back_porch) * (pixel_clocks_per_data_line + horizontal_front_porch + horizontal_back_porch))]

- 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

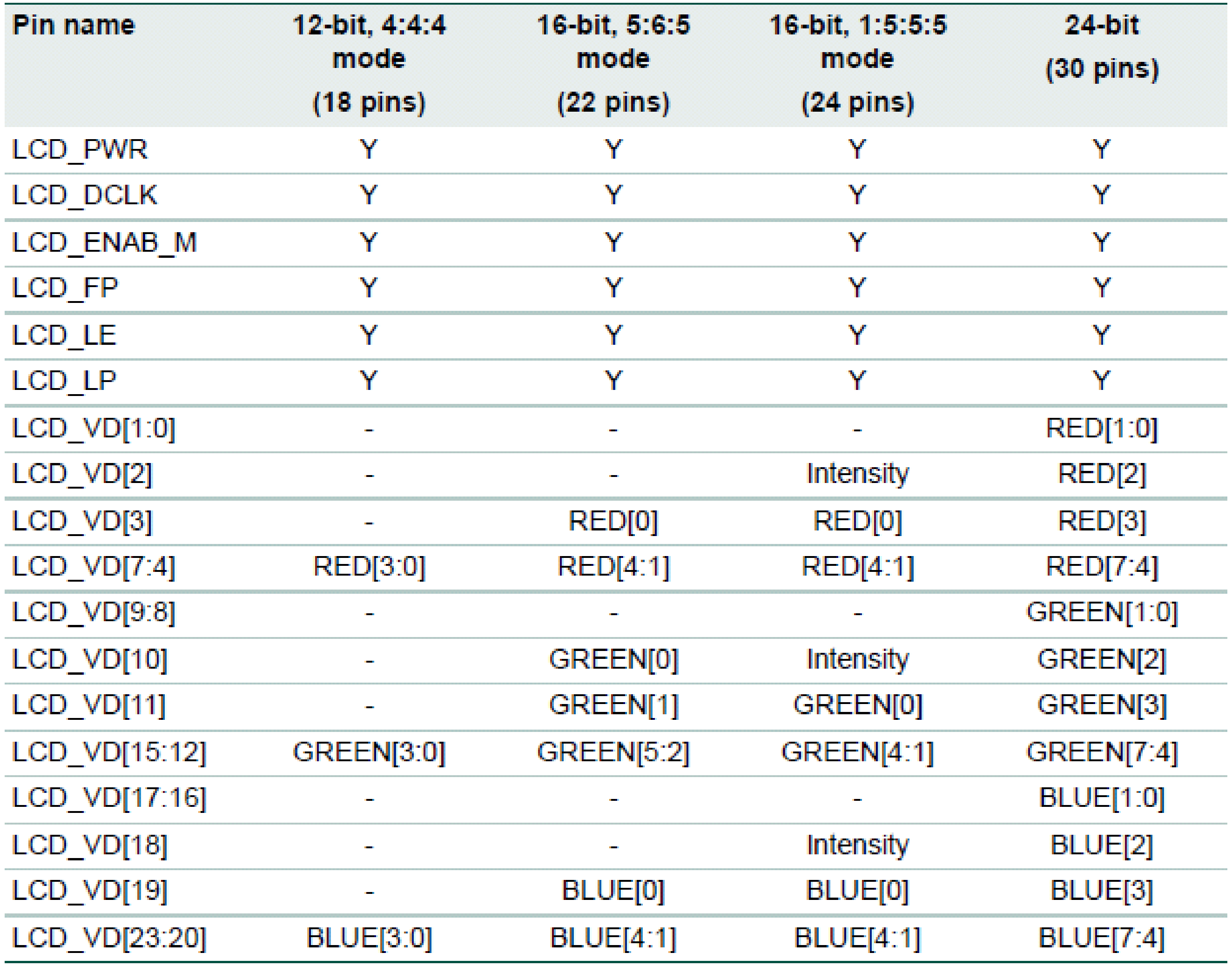
LCD Signals

- 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.



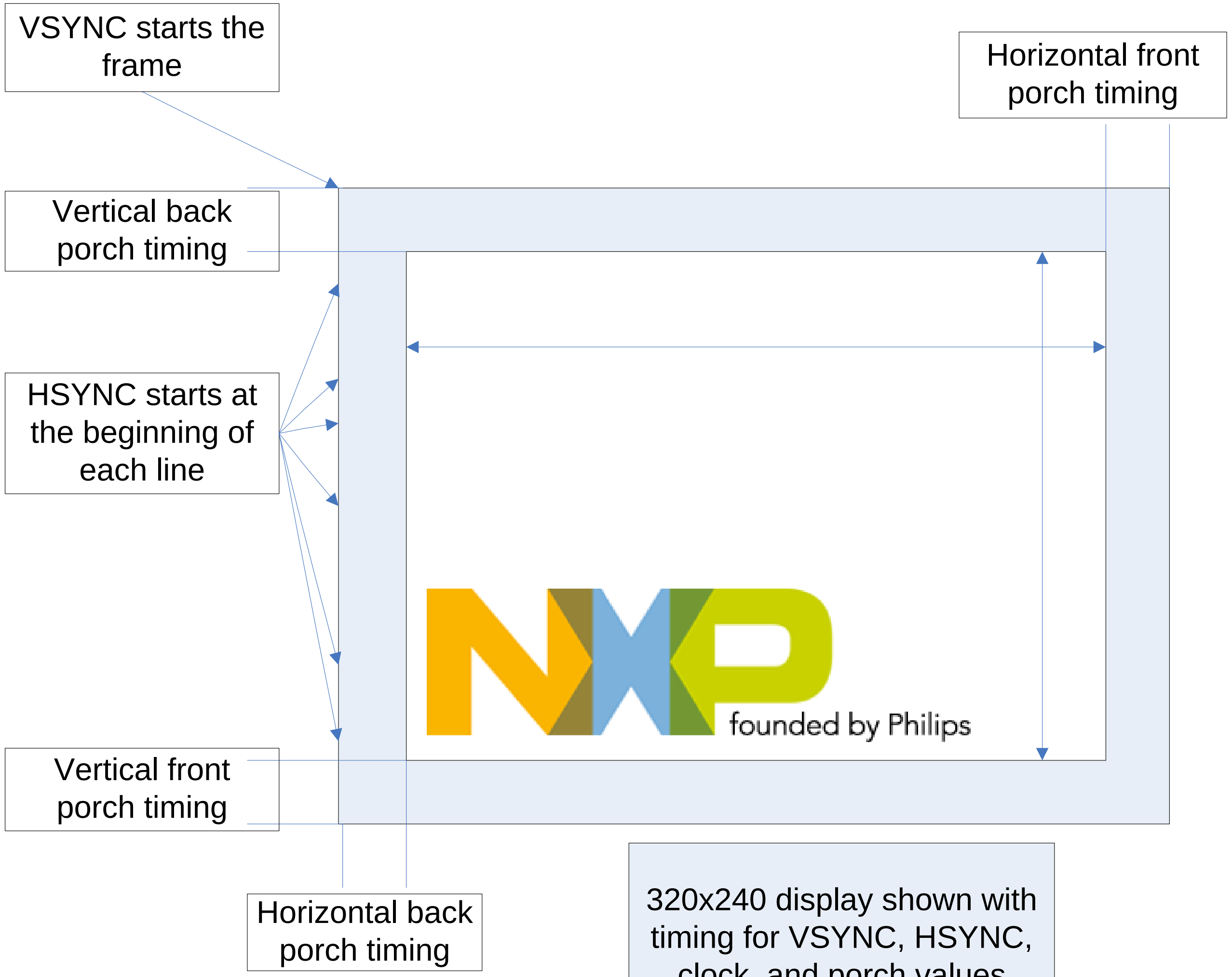
Pin name	Type	Function
LCD_PWR	output	LCD panel power enable.
LCD_DCLK	output	LCD panel clock.
LCD_ENAB_M	output	STN AC bias drive or TFT data enable output.
LCD_FP	output	Frame pulse (STN). Vertical synchronization pulse (TFT)
LCD_LE	output	Line end signal
LCD_LP	output	Line synchronization pulse (STN). 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



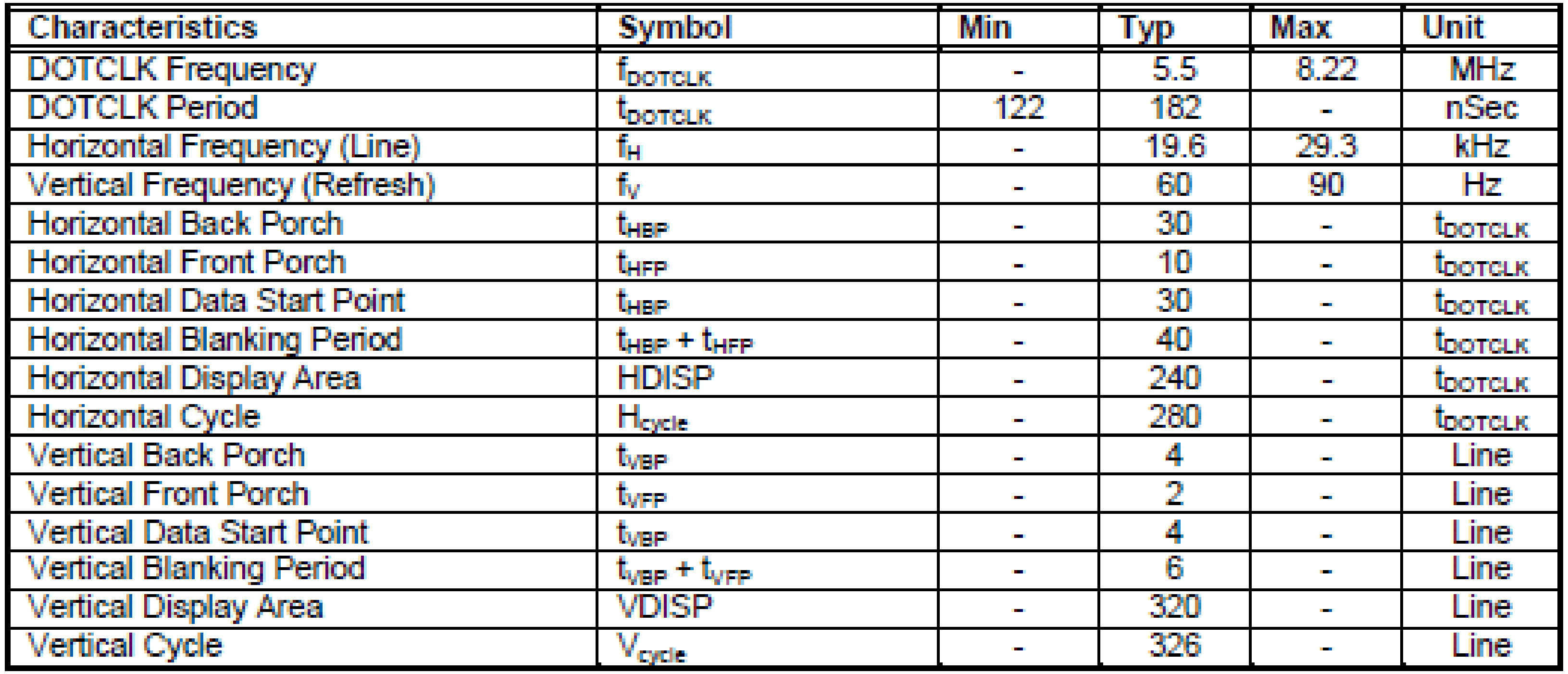
Pin name	12-bit, 4:4:4 mode (18 pins)	16-bit, 5:6:5 mode (22 pins)	16-bit, 1:5:5:5 mode (24 pins)	24-bit (30 pins)
LCD_PWR	Y	Y	Y	Y
LCD_DCLK	Y	Y	Y	Y
LCD_ENAB_M	Y	Y	Y	Y
LCD_FP	Y	Y	Y	Y
LCD_LE	Y	Y	Y	Y
LCD_LP	Y	Y	Y	Y
LCD_VD[1:0]	-	-	-	RED[1:0]
LCD_VD[2]	-	-	Intensity	RED[2]
LCD_VD[3]	-	RED[0]	RED[0]	RED[3]
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


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

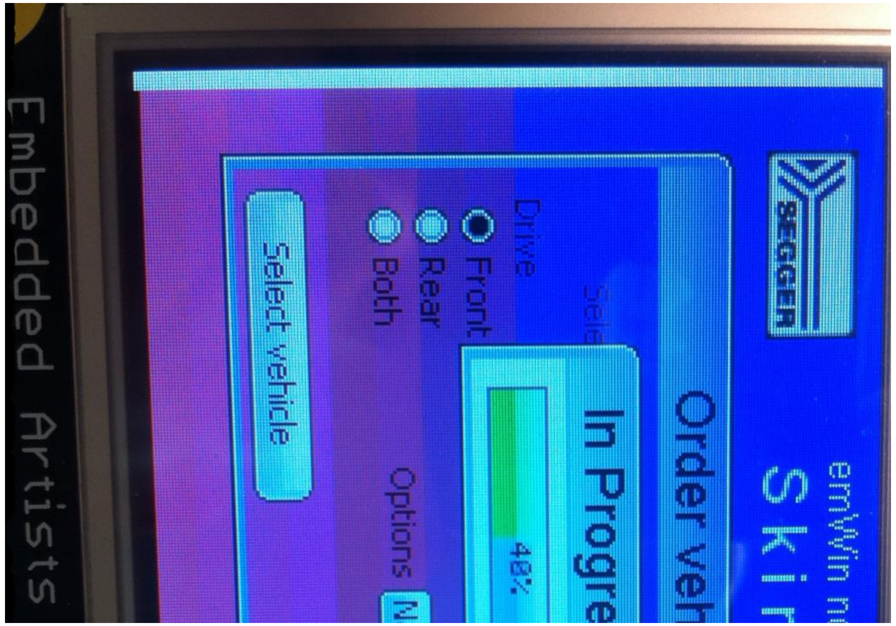
320x240 display shown with timing for VSYNC, HSYNC, clock, and porch values

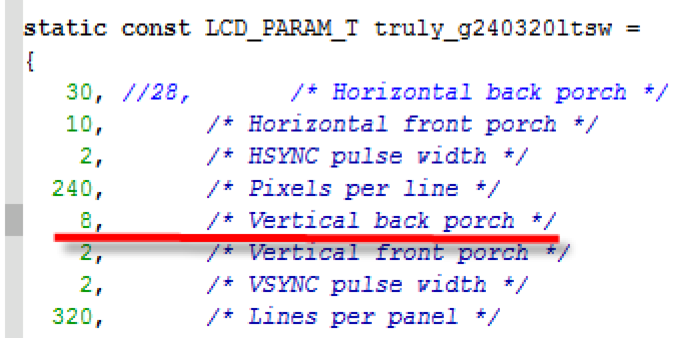
Example: Truly 240 x 320 TFT RGB666



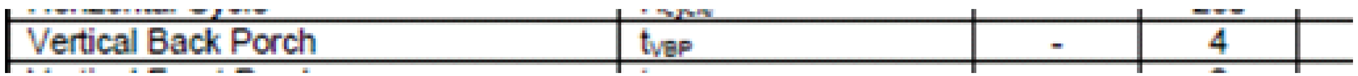
Characteristics	Symbol	Min	Typ	Max	Unit
DOTCLK Frequency	f_{DOTCLK}	-	5.5	8.22	MHz
DOTCLK Period	t_{DOTCLK}	122	182	-	nSec
Horizontal Frequency (Line)	f_H	-	19.6	29.3	kHz
Vertical Frequency (Refresh)	f_V	-	60	90	Hz
Horizontal Back Porch	t_{HBP}	-	30	-	t_{DOTCLK}
Horizontal Front Porch	t_{HFP}	-	10	-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

Snapshot of incorrect LCD settings





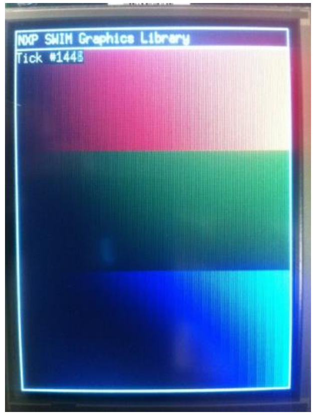
```
static const LCD_PARAM_T truly_g240320ltsw =
{
   30, //28,       /* Horizontal back porch */
   10,       /* Horizontal front porch */
    2,       /* HSYNC pulse width */
  240,       /* Pixels per line */
    8,       /* Vertical back porch */
    2,       /* Vertical front porch */
    2,   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

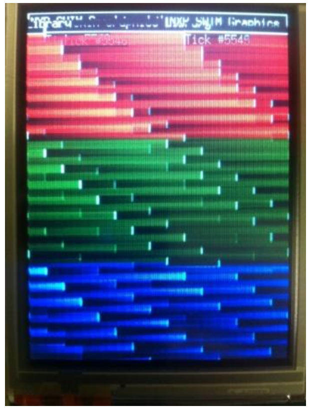


Vertical Back Porch	t_{VBP}	-	4

LCD Tearing

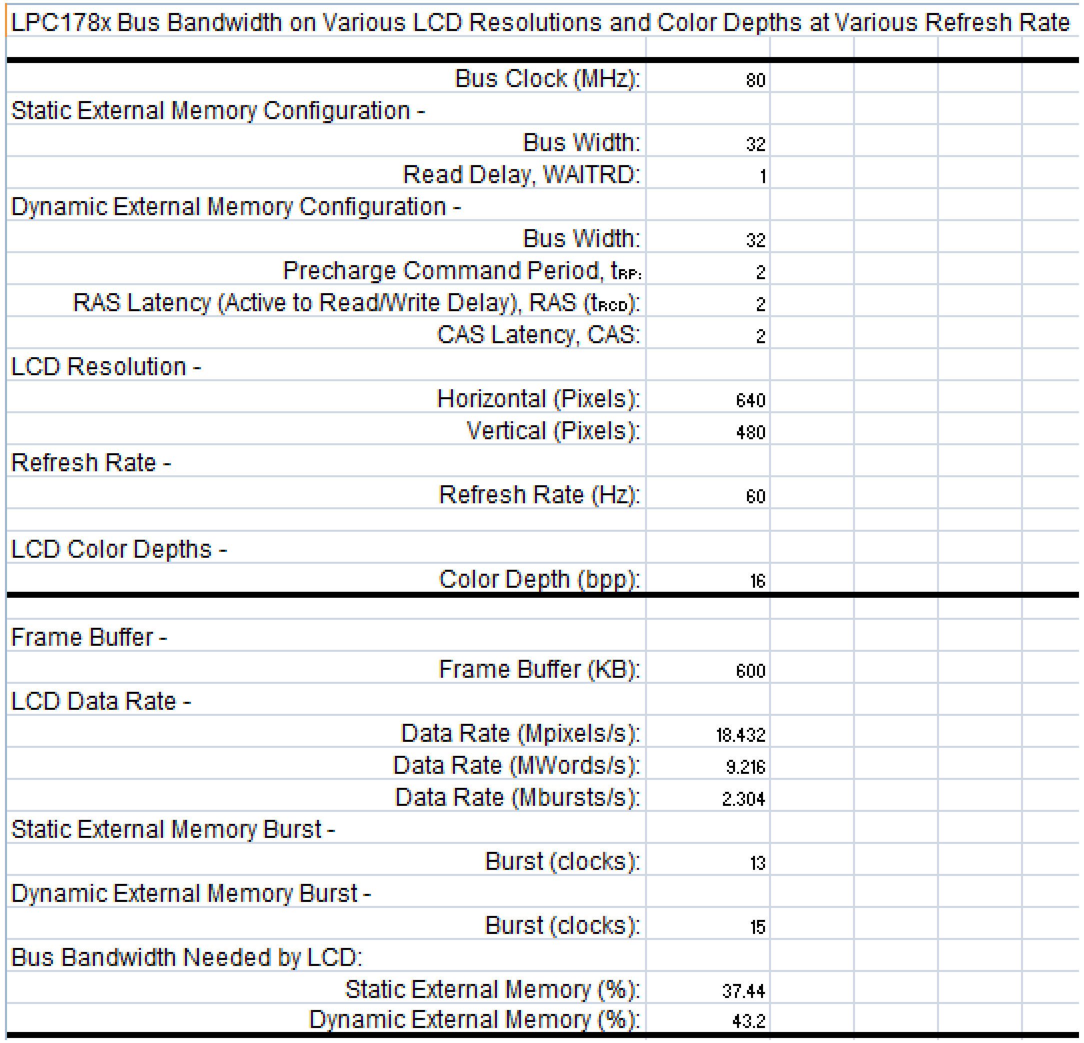
- Tearing:





- 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



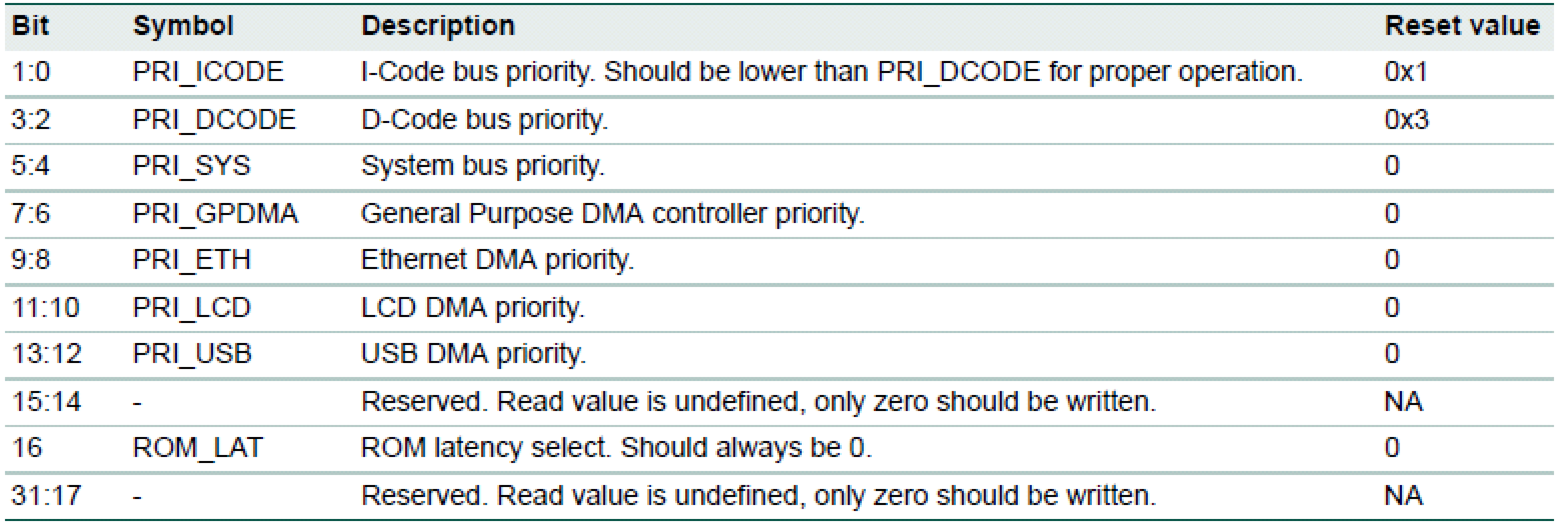
LPC178x Bus Bandwidth on Various LCD Resolutions and Color Depths at Various Refresh Rate

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_{RP}:	2
RAS Latency (Active to Read/Write Delay), RAS (t_{RCD}):	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.

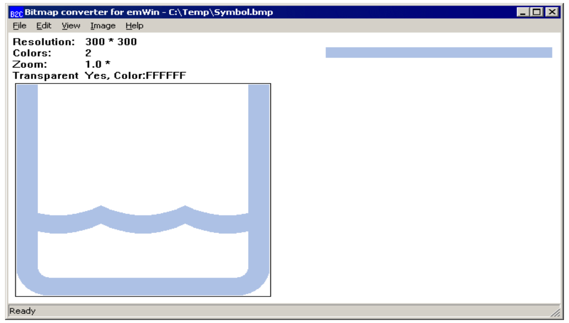
Bin2C - Can be used for converting any kind of file into byte arrays which then can be used within a C file.

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

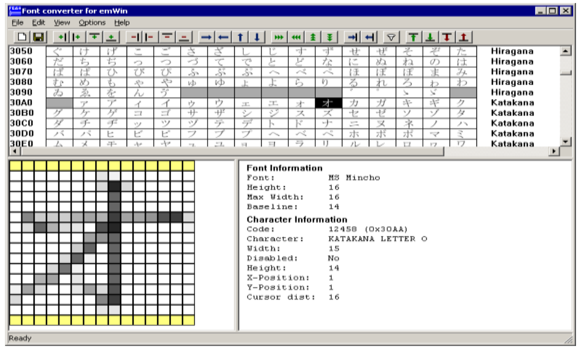
Optional tools are:

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.

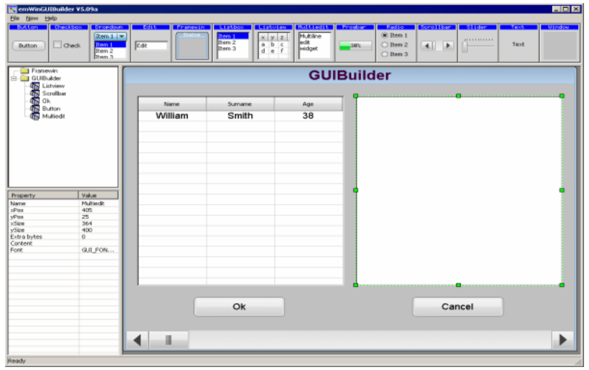

Bitmap converter for emWin - C:\Temp\Symbol.bmp
File Edit View Image Help
Ready

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



Font Information
Font: MS Mincho
Height: 16
Max Width: 16
Baseline: 14

Character Information
Code: 12458 (0x30AA)
Character: KATAKANA LETTER O
Width: 15
Disabled: No
Height: 14
X-Position: 1
Y-Position: 1
Cursor dist: 16



Name	Surname	Age
William	Smith	38

Property	Value
Name	Multiedit
xPos	405
yPos	25
xSize	364
ySize	400
Extra bytes	0
Content	
Font	GUI_FON...

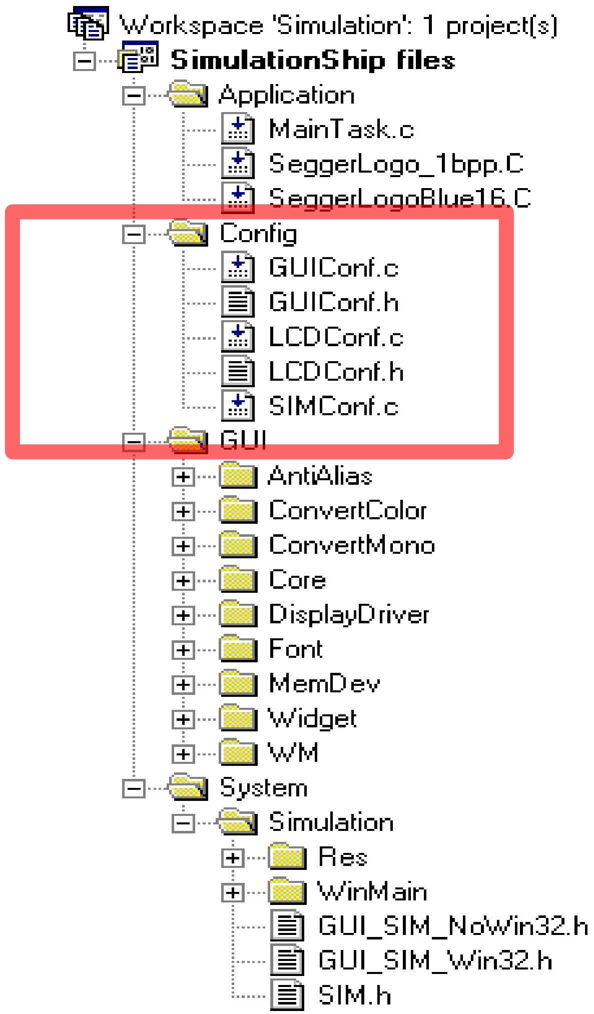
Configuration

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

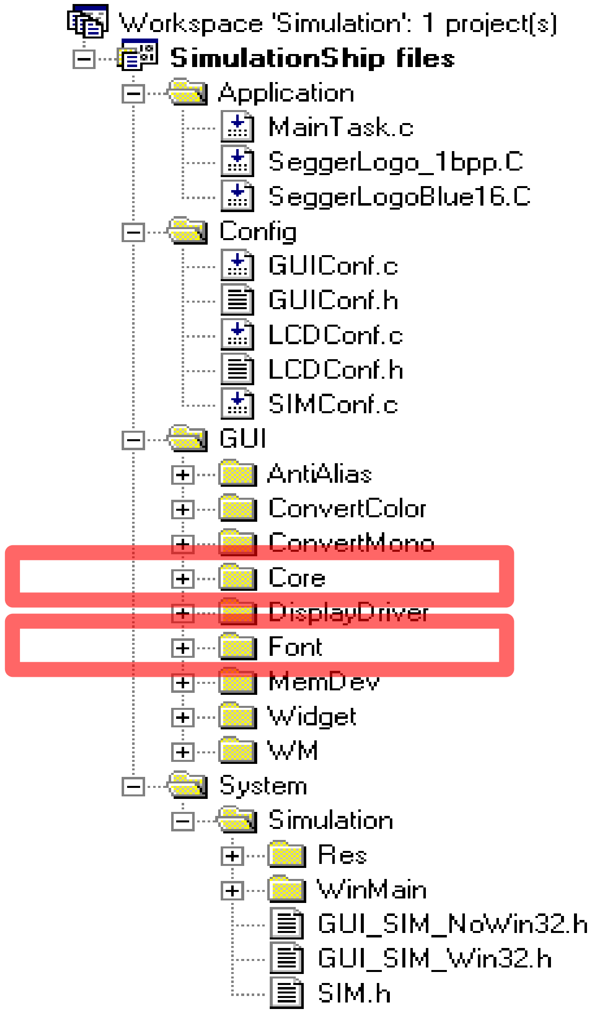


```
Workspace 'Simulation': 1 project(s)
  SimulationShip files
    Application
      MainTask.c
      SeggerLogo_1bpp.C
      SeggerLogoBlue16.C
    Config
      GUIConf.c
      GUIConf.h
      LCDConf.c
      LCDConf.h
      SIMConf.c
    GUI
      AntiAlias
      ConvertColor
      ConvertMono
      Core
      DisplayDriver
      Font
      MemDev
      Widget
      WM
    System
      Simulation
        Res
        WinMain
        GUI_SIM_NoWin32.h
        GUI_SIM_Win32.h
        SIM.h
```

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)**



```
Workspace 'Simulation': 1 project(s)
  SimulationShip files
    Application
      MainTask.c
      SeggerLogo_1bpp.C
      SeggerLogoBlue16.C
    Config
      GUIConf.c
      GUIConf.h
      LCDConf.c
      LCDConf.h
      SIMConf.c
    GUI
      AntiAlias
      ConvertColor
      ConvertMono
      Core
      DisplayDriver
      Font
      MemDev
      Widget
      WM
    System
      Simulation
        Res
        WinMain
        GUI_SIM_NoWin32.h
        GUI_SIM_Win32.h
        SIM.h
```

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

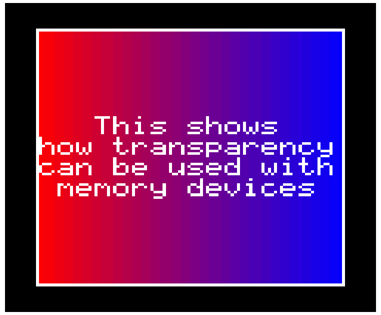
What means 'transparency' here?

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);
  }
}
```



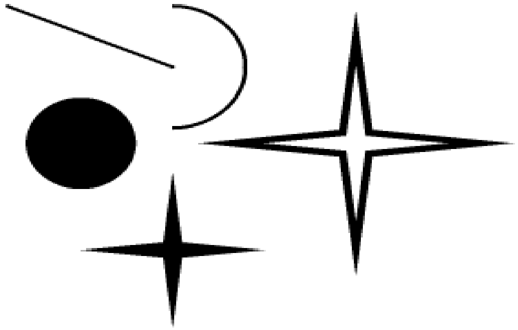
This shows
how transparency
can be used with
memory devices

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 **W**indow **M**anager?

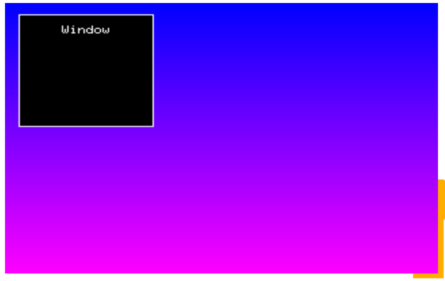
- **Management system for a hierarchic window structure**
  Each layer has its own desktop window. Each desktop window can have its own hierarchic tree of child windows.
- **Callback mechanism based system**
  Communication is based on an event driven callback mechanism. **All** drawing operations should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

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:

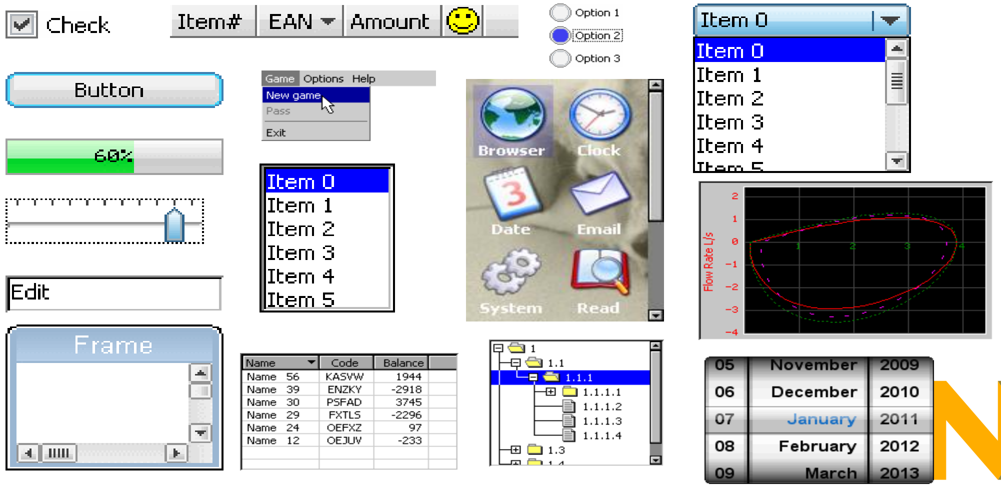
- **Direct creation**
  For each widget there exist creation functions:
  - `<WIDGET>_CreateEx()`
    Creation without user data.
  - `<WIDGET>_CreateUser()`
    Creation with user data.
- **Indirect creation**
  Indirect means using a dialog box creation function and a `GUI_WIDGET_CREATE_INFO` structure 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);
  }
}
```



Name	Code	Balance
Name 56	KASVW	1944
Name 39	ENZKY	-2918
Name 30	PSFAD	3745
Name 29	FXTLS	-2296
Name 24	OEFXZ	97
Name 12	OEJUV	-233

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:

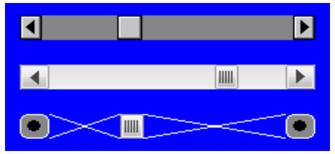
`<WIDGET>_SetDefaultSkin()`

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()`



```c
#include "DIALOG.h"

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

SAMPLE: Skinning\MainTask_ScrollBarSkin.c

GUI-Builder

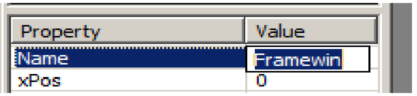
The GUI-Builder is a tool for creating dialogs without any knowledge of the C programming language.

Basic usage of the GUI-Builder:

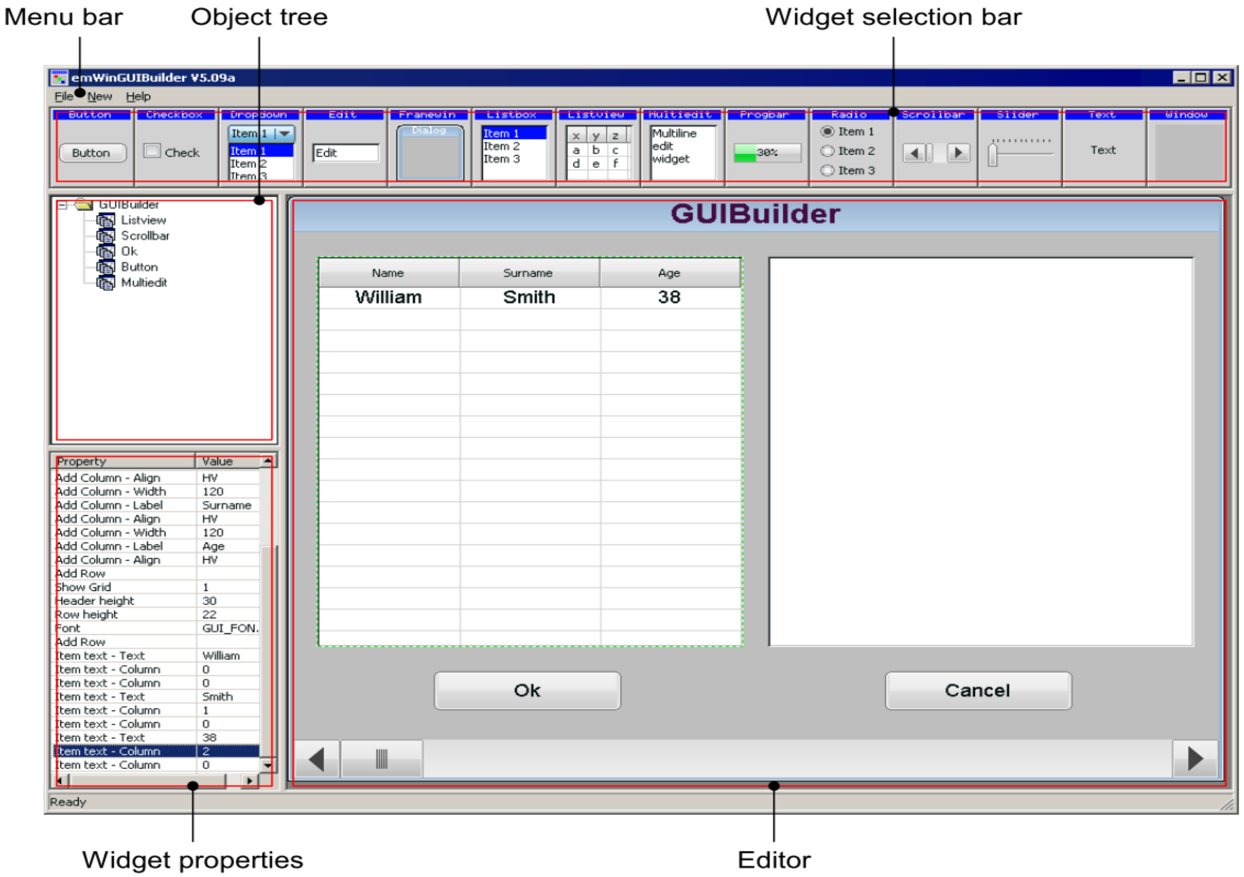
- **Check project path in** `GUIBuilder.ini`
  This file can be found in the application folder of the tool.
- **Start GUI-Builder**
- **Start with** `FRAMEWIN` **or** `WINDOW` **widget**
  Only these widgets are able to serve as parent windows for a dialog here.
- **Place widgets within the parent window**
  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



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**

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

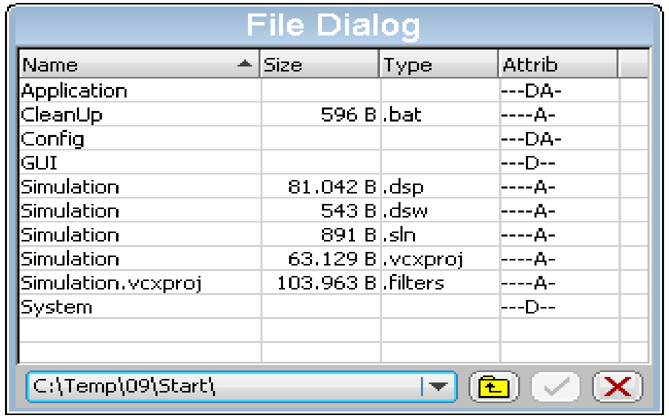
- Embedded file system explorer.
- Simple callback mechanism used to get file data from application.
- Ready to use sample for emFile available

CHOOSECOLOR

- Selection from an application defined array of colors.

CALENDAR

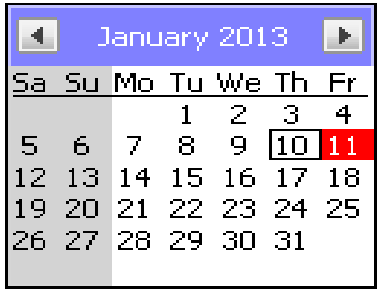
- Selection of a date of the gregorian calendar.



File Dialog

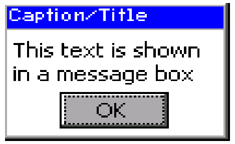
Name	Size	Type	Attrib
Application			---DA-
CleanUp	596 B	.bat	----A-
Config			---DA-
GUI			---D--
Simulation	81.042 B	.dsp	----A-
Simulation	543 B	.dsw	----A-
Simulation	891 B	.sln	----A-
Simulation	63.129 B	.vcxproj	----A-
Simulation.vcxproj	103.963 B	.filters	----A-
System			---D--

C:\Temp\09\Start\



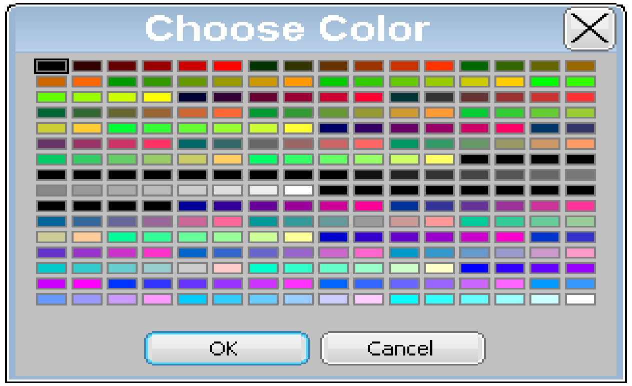
January 2013

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	



Caption/Title

This text is shown in a message box



Choose Color

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 µVision / 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

HANDS-ON SESSION 2

Hands-on Session 2 Objectives

1. Build and run emWin example from the lpcopen package
2. Working with Touch, Cursors and Fonts
3. Create a new GUI using emWin tools
4. Add USB Device functionality to your graphic application

Step by step handouts to be provided, with additional details.

Helps

- As you progress through this lab, use your pencil to check off each heading or bullet as you complete it. Mark any bullet with a '?' if you have a question.
- LPCXpresso IDE
  - When given the instruction to Build and Debug the project, always be sure to Terminate the current debug session, or the IDE will throw errors.
  - Highlight any symbol, right-click and choose Open Declaration will find its original value or declaration from anywhere in the project.
  - When modifying code, look for a "ToDo" symbol in the LPCXpresso editor, and you will see where to make the changes.
- We will make frequent reference to the emWin User Guide. Open this document from the workshop's Documentation folder, and make use of the Search function to find needed API calls or definitions.
- Hardware:
  - Capacitive touch screens require a lot of tuning; in case there is a touch glitch please don't be alarmed. A calibration has been implemented which should work well for most displays.

Code walkthrough

For the default *DIALOG_SliderColor* application, here are some of the important sections of code to preview. Stay at high level, just to know the idea of what is the function’s basic job.

- main()
- main->prvSetupHardware
- main->prvSetupHardware->disp_init()
- main->xTaskCreate()->vGUITaskEmWin()
- main->vGUITaskEmWin()->emwinhal_init()
- main->vGUITaskEmWin()->MainTask()
- main->vGUITaskEmWin()->MainTask()->GUI_Init()
- main->vGUITaskEmWin()->MainTask()->GUI_ExecDialogBox()

SECURE CONNECTIONS
FOR A SMARTER WORLD