STM32F429 Seminar

High-performance Cortex™-M4 MCU

life.augmented

- Hands-On
  - Ensure you picked-up
    - USB Flash Drive with STM32F429 Discovery Contents
    - USB Cable
    - STM32F429 Discovery Kit – will be provided after software is loaded

Agenda

Presentation

Time	
9:00 – 12:00	• Introduction to the STM32 F4 High Performance MCU Family • STM32F429 Advanced System Architecture including ART™ Memory Accelerator and Chrom-ART Graphics Accelerator • Hands-On Session: Graphics with the Chrom-ART Graphics Accelerator
12:00 – 1:00	Lunch
1:00 – 3:00	• STM32F429 Smart Digital and Analog Peripherals • Hands-On Session: DSP and FPU performance with a 180 MHz Cortex-M4 • Hands-On Session: Application example using the L3GD20 three axis digital gyroscope and color QVGA display • STM32 Software Development Suites • Hands-On Session: MicroXplorer Graphical Configuration Tool

IAR Tool Installation

Systems Check

- Everyone should have
  - A Windows ® Laptop (Windows 7 or Windows 8)
  - ST Provided 8GB USB Flash Drive
  - USB Cable
  - STM32F429I-Discovery Kit (will be provided after driver installation)

- Ready to begin?

Step #1 - File Installation

- Insert the USB Flash Drive into your Laptop
- Copy the folder **“…\STM32Seminar”** on the USB flash drive to your root **“c:\”** folder
  - **C:\STM32Seminar**
    - **Edit folder properties and remove ‘Read-only’ attribute for all sub-folders.**
- Open this directory and you will find the following:
  - C:\STM32Seminar\IAR_EWARM
    - IAR tool installation application and license file.
    - ST-LINK/V2 Windows driver installer.

Step #2 – EWARM Installation

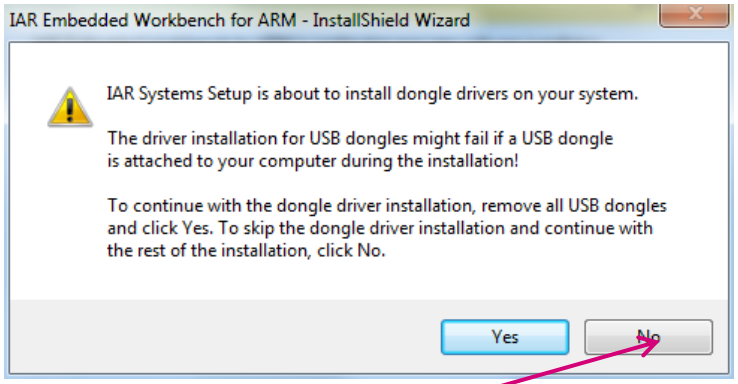
- For this workshop, we will be using the full version of the IAR Embedded Workbench for ARM. The license you are about to install will be valid for 5 days.
- Browse to → C:\STM32Seminar\IAR_EWARM and Double-click on the file ***EWARM-CD-6602-5507.exe*** to begin the IAR installation.
  - Click through the default options and accept the 'license agreement' when prompted.
  - If you have an existing installation of IAR you should install in a new directory
- Please ask any of the proctors for assistance if you have an issue.

Step #3 - EWARM Installation

Single-click on *Install IAR Embedded Workbench®* **to begin the installation.**

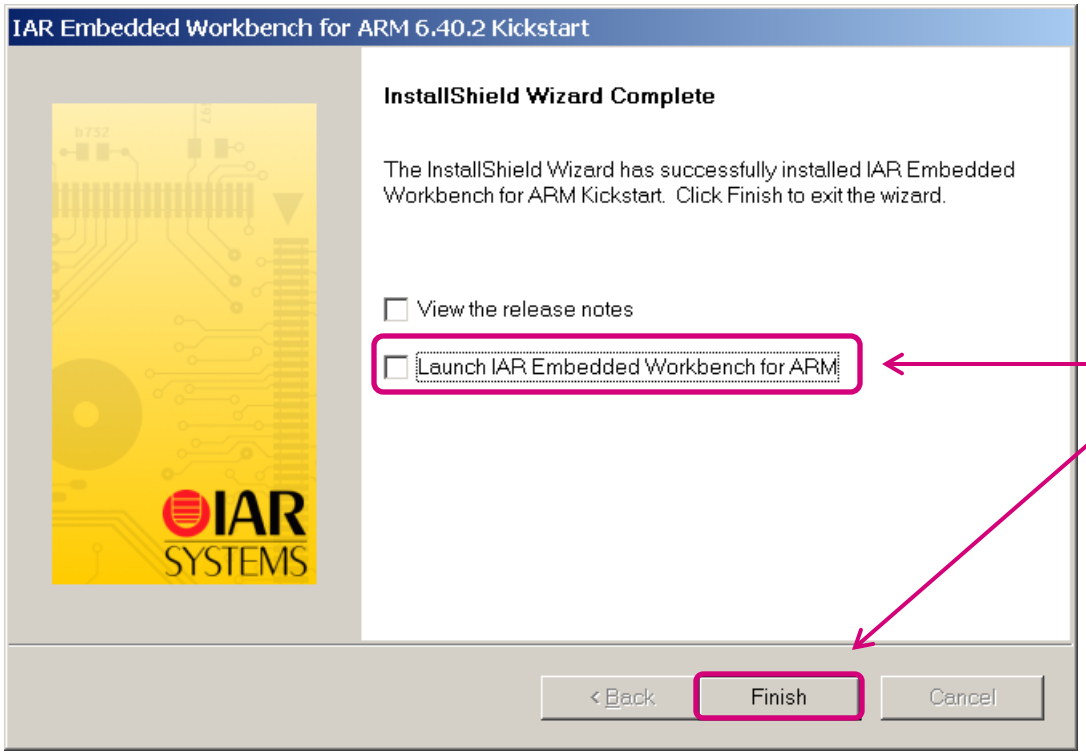
Step #4 - EWARM Installation

Make sure your Discovery Kit is not connected to USB

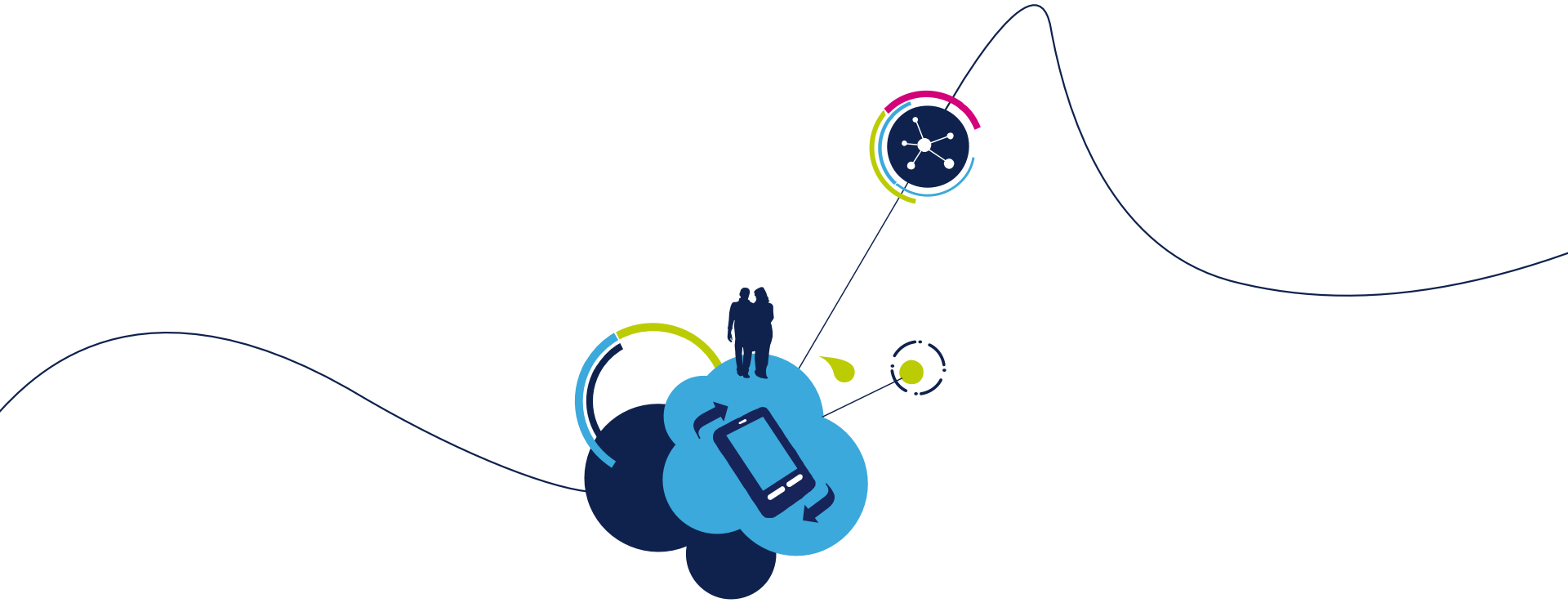


Single-click on *NO* to skip installing the dongle drivers on your system.

Step #4 - EWARM Installation



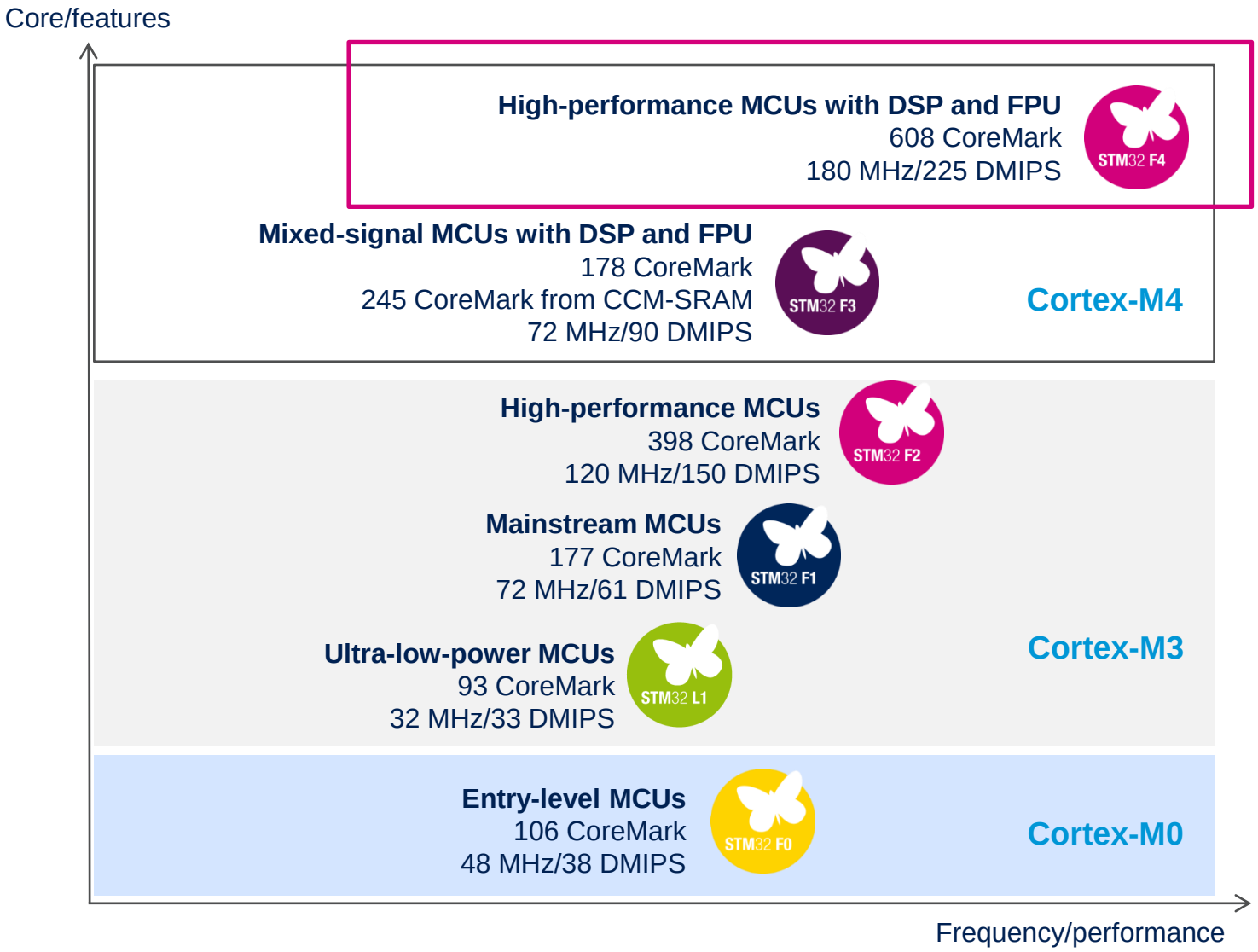
Installation is complete. **'Uncheck'** both switches. select **'Finish'**, and wait to proceed to the next step (Total time ~ 25 min).



Overview of the STM32 portfolio

life.augmented

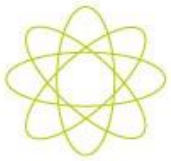
Today - STM32 portfolio overview



life.augmented

5 reasons to chose an STM32

Real-time performance



ART Accelerator™, Chrom-ART Accelerator™, CCM-SRAM,
Multi-AHB bus matrix,
Excellent real-time up to 180 MHz/ 225 DMIPS
Zero-wait state execution performance from Flash

Outstanding power efficiency

< 1 µA RTC in V_{BAT} mode, ultra-low dynamic power consumption
140 µA/MHz
1.65 to 3.6 V V_{DD},
0.45 µA Stop mode and 0.3 µA Standy mode

Superior and innovative peripherals

USB-OTG High speed, Ethernet, CAN, LCD-TFT controller, crypto/hash processor, PGA, 16-bit ∑Δ ADC and 12-bit ADC (up to 5 MSPS), external memory interface, CEC

Maximum integration

Reset circuitry, voltage regulator, internal RC oscillator, PLL, WLCSP packages

Extensive ecosystem

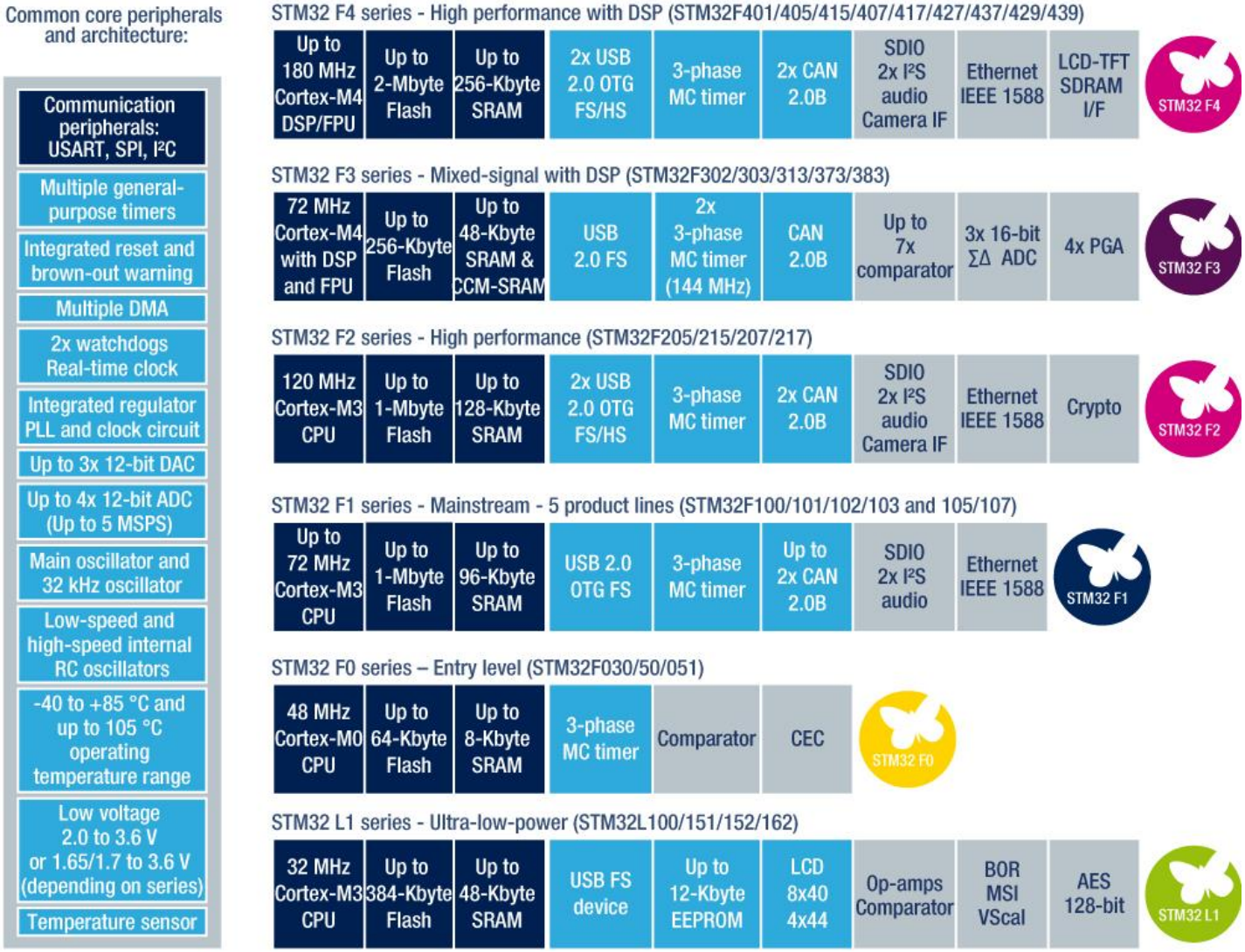
ARM + ST ecosystem (eval boards, discovery kits, software libraries, RTOS)

More than **450 compatible devices**

Releasing your creativity

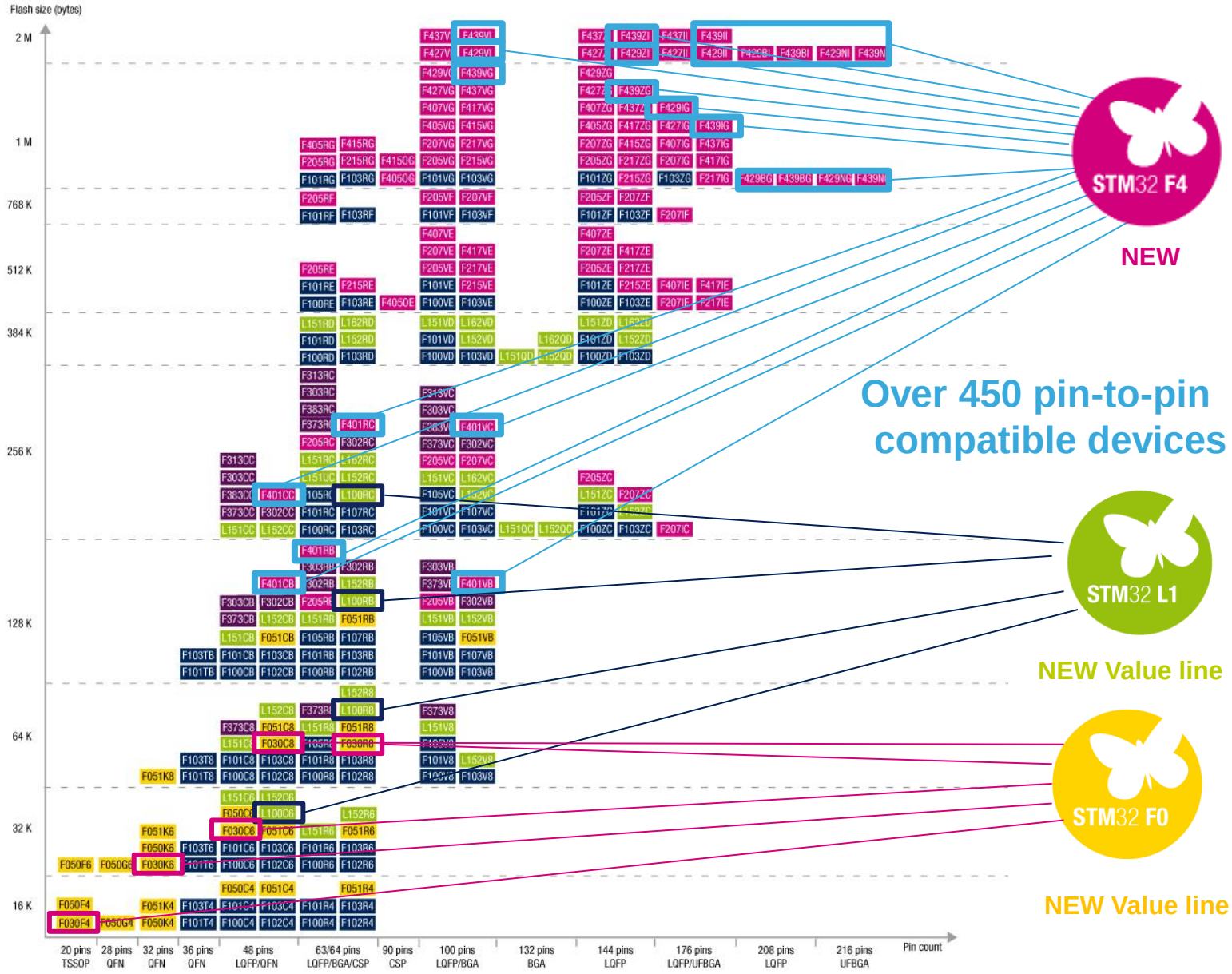
life.augmented

STM32 – 6 product series



life.augmented

STM32 – leading Cortex-M portfolio



life.augmented

ST has licensed all Cortex-M processors

- **Forget traditional 8/16/32-bit classifications and get**
  - Seamless architecture across all applications
  - Every product optimized for ultra-low power and ease of use

Cortex-M0	Cortex-M3	Cortex-M4
8/16-bit applications	16/32-bit applications	32-bit/DSC applications

Binary and tool compatible

life.augmented

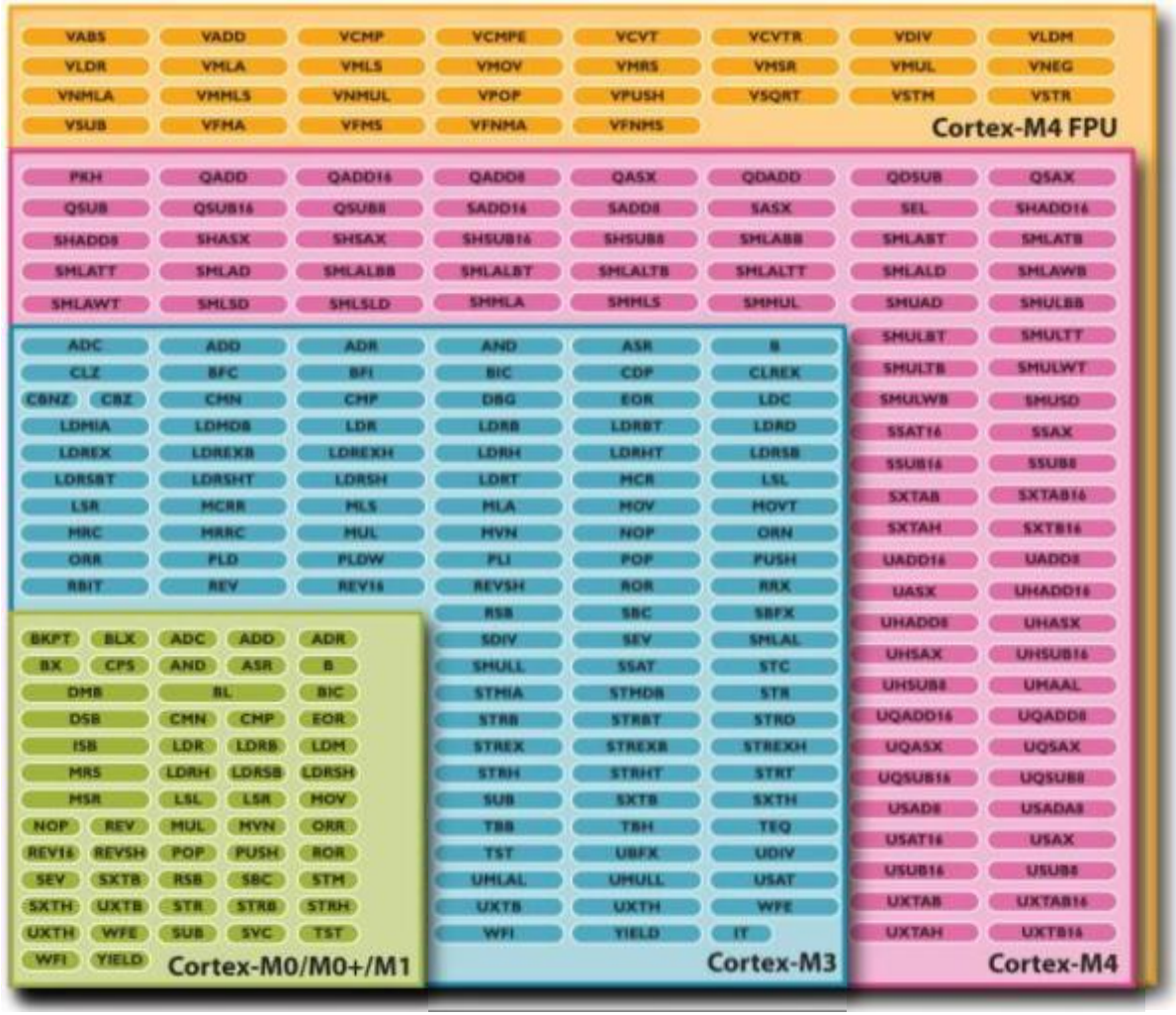
Cortex-M processors binary compatible

Floating Point Unit (FPU)

DSP (SIMD, fast MAC)

Advanced data processing
Bit field manipulations

General data processing
I/O control tasks



Source: ARM

A large community of partners

Hardware Development Tools

- Promotion Kits
  - Discovery Kits

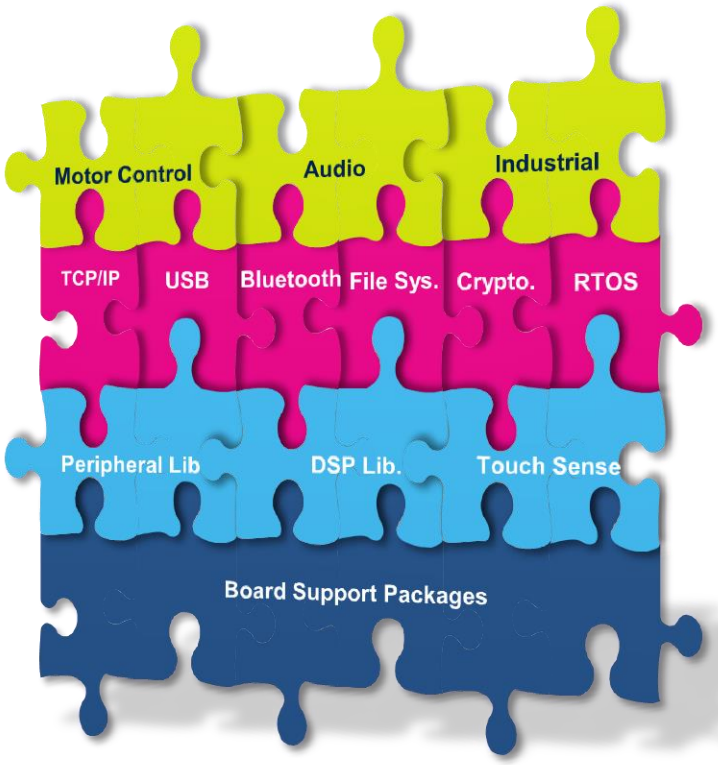
- Evaluation Boards

- Open Hardware Boards
  - Arduino-based
    - Leaflabs Maple, Olimexino-STM32, Netduino,…
  - Microsoft Gadgeteer-based
    - Netduino Go, Mountaineer, GHI…

- Debug Probes and Programming Tools
  - ST-Link
  - I-jet and JTAGjet-Trace
  - Ulink

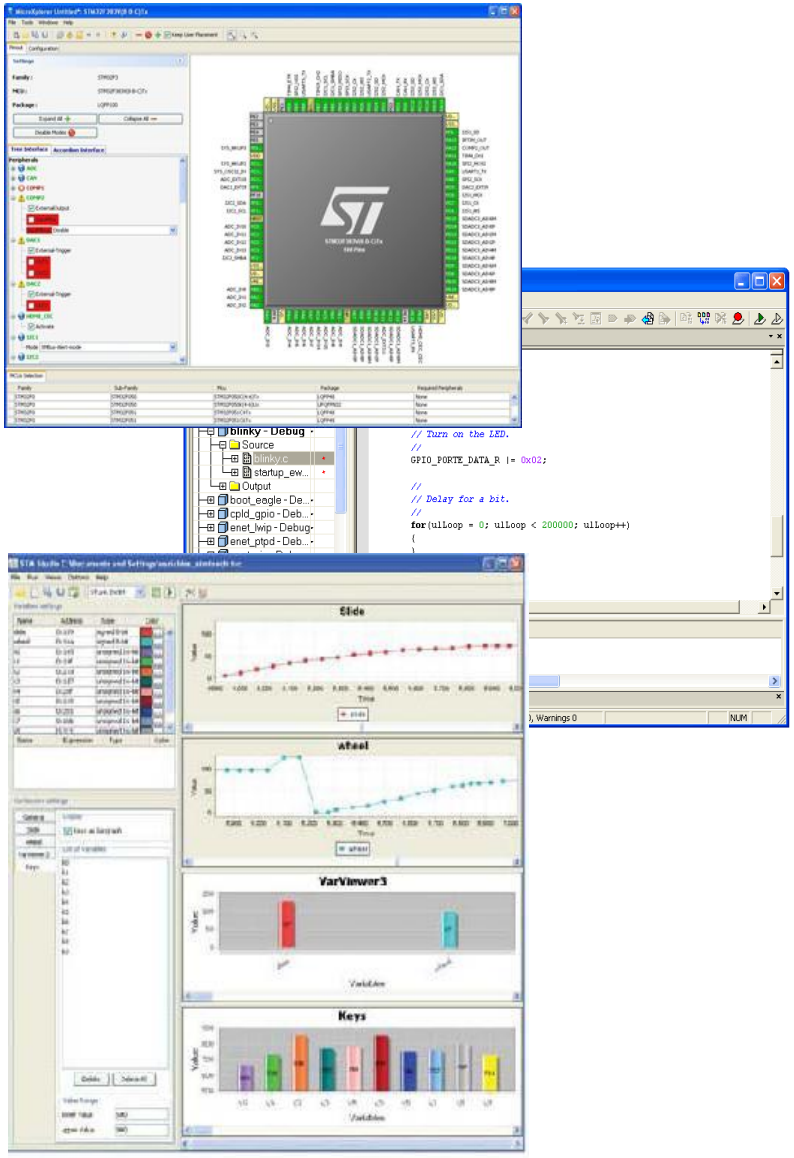
Embedded Software (Firmware)

- HAL / Drivers
  - ST Boards Support Packages (BSP)
  - Peripheral Libraries (Drivers)
  - DSP Library
- RTOS / Firmware Stacks
  - RTOS
  - Cryptographic
  - USB
  - TCP/IP
  - File Systems
  - BlueTooth
  - Graphics
  - Touch sensing
- Application Bricks
  - Audio
  - Industrial
  - Motor Control
- High Level Frameworks (STM32 only)
  - Java
  - Microsoft .Net Micro Framework
  - Matlab/Simulink



Software Development Tools

- Configuration Tools
  - MicroXplorer
- Development and Debugging Tools
  - IAR EWARM
  - Keil MDK
  - Atollic TrueStudio
  - Rowley CrossWorks
  - Emprog ThunderBench
  - Embest CooCox
  - Segger emIDE
  - Raisonance Ride
  - Altium Tasking
  - Cosmic Idea
  - Yagarto…
- Monitoring Tools
  - STMStudio



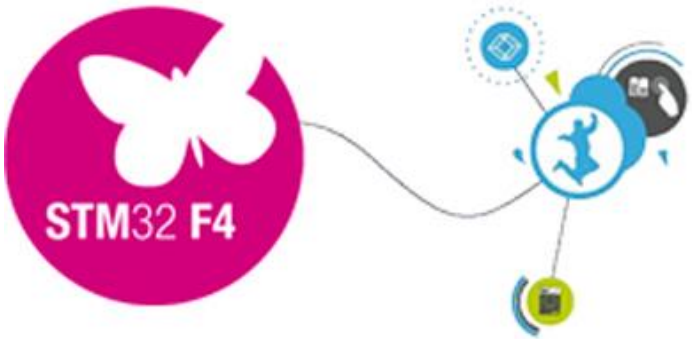
NEW STM32F429

High-performance Cortex™-M4 MCU

life.augmented

High-performance MCUs with DSP and FPU

- **World's highest performance Cortex-M MCU** executing from Embedded Flash, Cortex-M4 core with FPU up to 180 MHz/225 DMIPS
- **High integration** thanks to ST 90nm process (same platform as F2 series): up to 2MB Flash/256kB SRAM
- **Advanced features and connectivity** Chrom-ART Graphics Accelerator, USB OTG, Ethernet, CAN, SDRAM interface, LCD TFT controller
- **Power efficiency** thanks to ST90nm Process and voltage scaling



ARM Cortex™-M4 Core

FPU

- Single precision
- Ease of use
- Better code efficiency
- Faster time to market
- Eliminate scaling and saturation
- Easier support for meta-language tools

MCU

- Ease of use of C programming
- Interrupt handling
- Ultra-low power

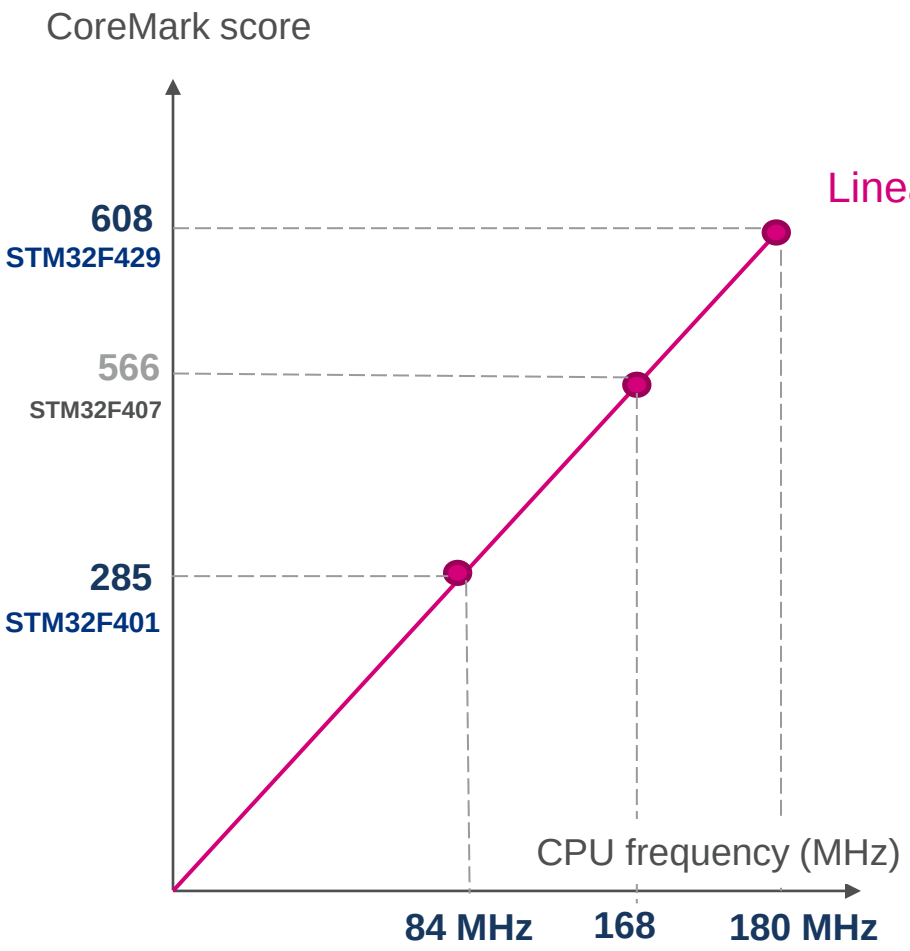
DSP

- Harvard architecture
- Single-cycle MAC
- Barrel shifter

STM32F429 Highlights

- 180 MHz/225 DMIPS
- Dual bank Flash (in both 1-MB and 2-MB), 256kB SRAM
- SDRAM Interface (up to 32-bit)
- LCD-TFT controller supporting up to SVGA (800x600)
- Better graphics with ST **Chrom-ART Accelerator™**:
  - **x2 more performance vs. CPU alone**
  - **Offloads the CPU for graphical data generation:**
    - Raw data copy
    - Pixel format conversion
    - Image blending (image mixing with some transparency)
- 100 µA typ. in Stop mode

Providing **more performance**

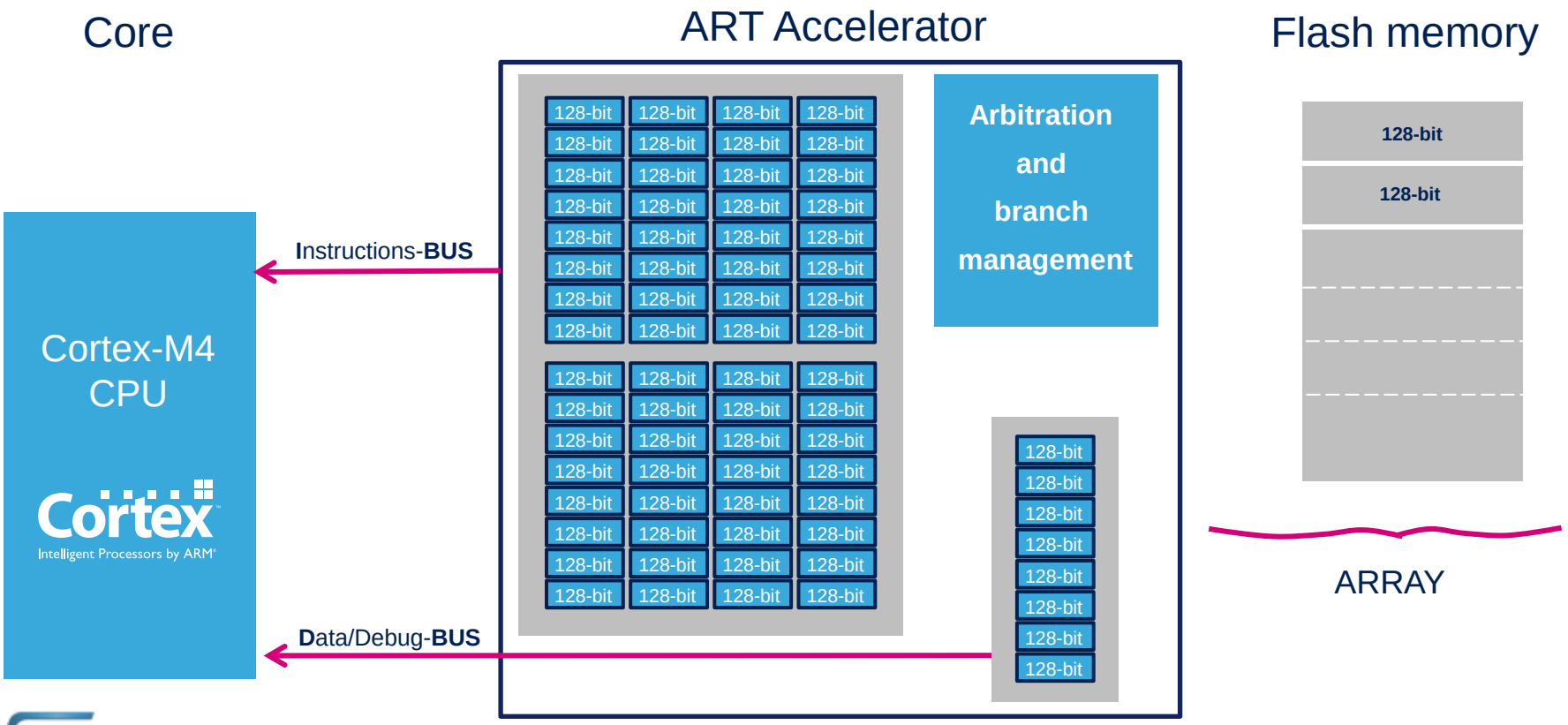


Linear **execution performance** from Flash

- **Up to 180** MHz/ 225 DMIPS with ART Accelerator™
- **Up to 608** CoreMark Result
- ARM Cortex-M4 with floating-point unit (FPU)

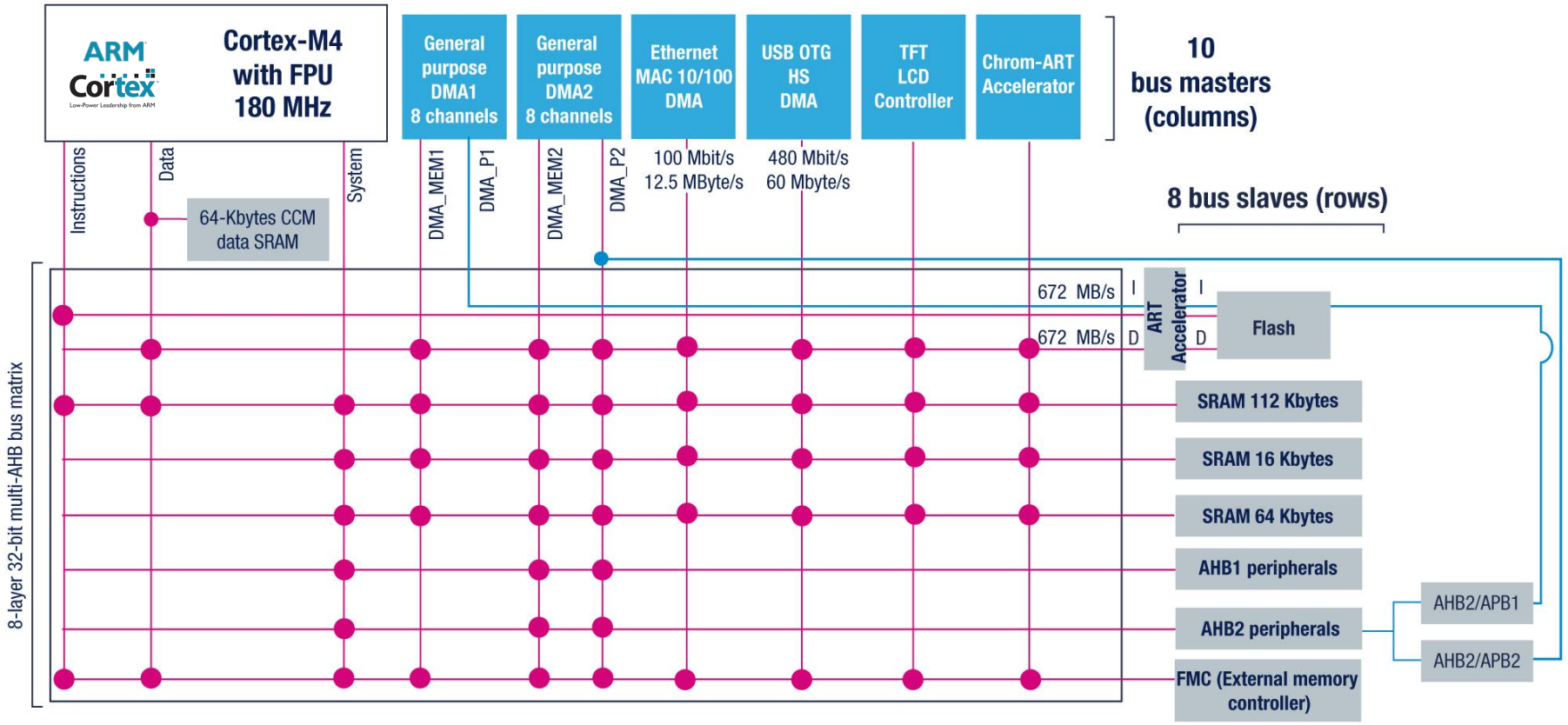
ST ART Accelerator for Processing performance

- The ART (Adaptive Real-Time) memory accelerator unleashes processing performance equivalent to 0-wait state Flash execution up to 180 MHz for F4 series



System performance

32-bit multi-AHB bus matrix



Advanced connectivity

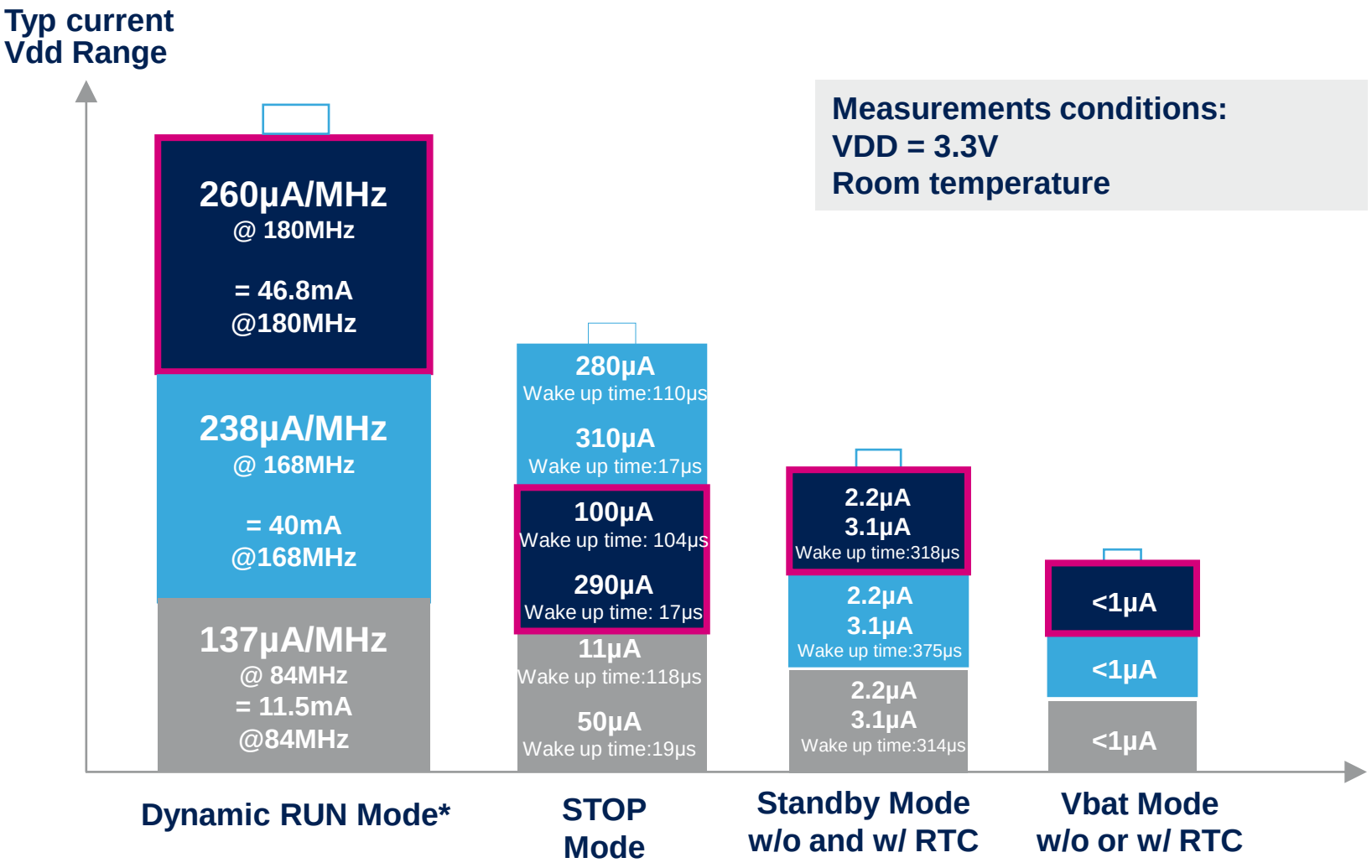
Peripherals	Performance
USB FS / HS	12 Mbit/s / 480 Mbit/s
USART	Up to 11.25 Mbit/s
SPI	Up to 45 Mbit/s
I²C	1Mbit/s
GPIO toggling	Up to 90 MHz
3-phase MC timer	180 MHz PWM timer clock input
SDIO	Up to 48 MHz
I²S and SAI	From 8 kHz to 192 kHz sampling frequencies
Camera interface	Up to 54 Mbyte/s at 54 MHz (8- to 14-bit parallel)
Crypto/hash processor	AES-256 up to 149.33 Mbyte/s
FMC	Up to 90 MHz (8-/16-/32-bit data bus, supports SRAM, PSRAM, NAND and NOR Flash, SDRAM, parallel graphic LCD)
12-bit ADC / 12-bit DAC	0.41 µs (2.4 MSPS, 7.2 MSPS in Interleaved mode) / 1 MSPS dual DAC
CAN 2.0B	Up to 2 independent CAN
Ethernet	10/100 Mbit/s MAC with hardware IEEE 1588
LCD TFT controller	Display size : QVGA, QWVGA, VGA, SVGA with 2-layer blending and dithering

STM32F4
power efficiency

- Outstanding dynamic power consumption thanks to ST 90nm process

- Low leakage current made possible by advanced design technics and architecture (voltage scaling)

Power consumption efficiency



Legend:

* Run mode Conditions: Coremark executed from Flash, peripherals OFF

STM32F427/437 and STM32F429/439 STM32F405/415 and STM32F407/417 STM32F401

High integration

Series	STM32 High-performance platform	
Flash/SRAM (bytes)	**STM32F2**	**STM32F4**
256 K/64 K		WLCSP49 (3x3mm) F401 UFBGA100 (7x7mm) UFQFPN48 (7x7mm)
512 K/128 K		WLCSP56 (3.35x3.35mm) F401 UFBGA100 (7x7mm)
1 M/128 K	WLSCP66 F205 (<3.7x4mm)	
1 M/192 K		WLSCP90 F405 (<4x4.3 mm)
2 M/256 K		WLSCP143 F429 (<4.5x5.6mm)

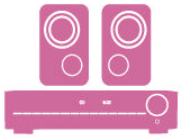
STM32F4 multiple applications

Industrial

- PLC
- Inverters
- Power meters
- Printers, scanners
- Industrial networking
- Industrial motor drive
- Communication gateway

Building & security

- Alarm systems
- Access control
- HVAC



Consumer

- PC peripherals, gaming
- Digital cameras, GPS platforms
- Home audio
- Wi-Fi , Bluetooth modules
- Smartphone accessories

Medical

- High-end glucose meters
- Power meters
- Battery-operated applications

life.augmented

Note:
1/ 1.7 V min on specific packages
2/ Hardware crypto/hash on F415/417 and F437/439 only

STM32 F4 product lines

Main common features

Cortex™-M4 (DSP + FPU)

- Up to 2x USB 2.0 OTG FS/HS
- SDIO
- USART, SPI, I²C
- I²S + audio PLL
- 16- and 32-bit timers

- Up to 3x 12-bit ADC (0.41 µs)

- Low voltage 1.7[1] to 3.6 V

STM32F429/439

180 MHz 1 to 2-MB Flash 256-KB SRAM	Hardware Crypto /hash² RNG	2x 12-bit DAC	Ethernet IEEE 1588 2x CAN	Camera interface	SDRAM interface FMC	Serial audio interface (SAI)	Chrom-ART™ Accelerator	TFT LCD controller

STM32F427/437

180 MHz 1 to 2-MB Flash 256-KB SRAM	Hardware Crypto /hash² RNG	2x 12-bit DAC	Ethernet IEEE 1588 2x CAN	Camera interface	SDRAM interface FMC	Serial audio interface (SAI)	Chrom-ART™ Accelerator

STM32F407/417

168 MHz 512-KB to 1-MB Flash 192-KB SRAM	Hardware Crypto /hash² RNG	2x 12-bit DAC	Ethernet IEEE 1588 2x CAN	Camera interface

STM32F405/415

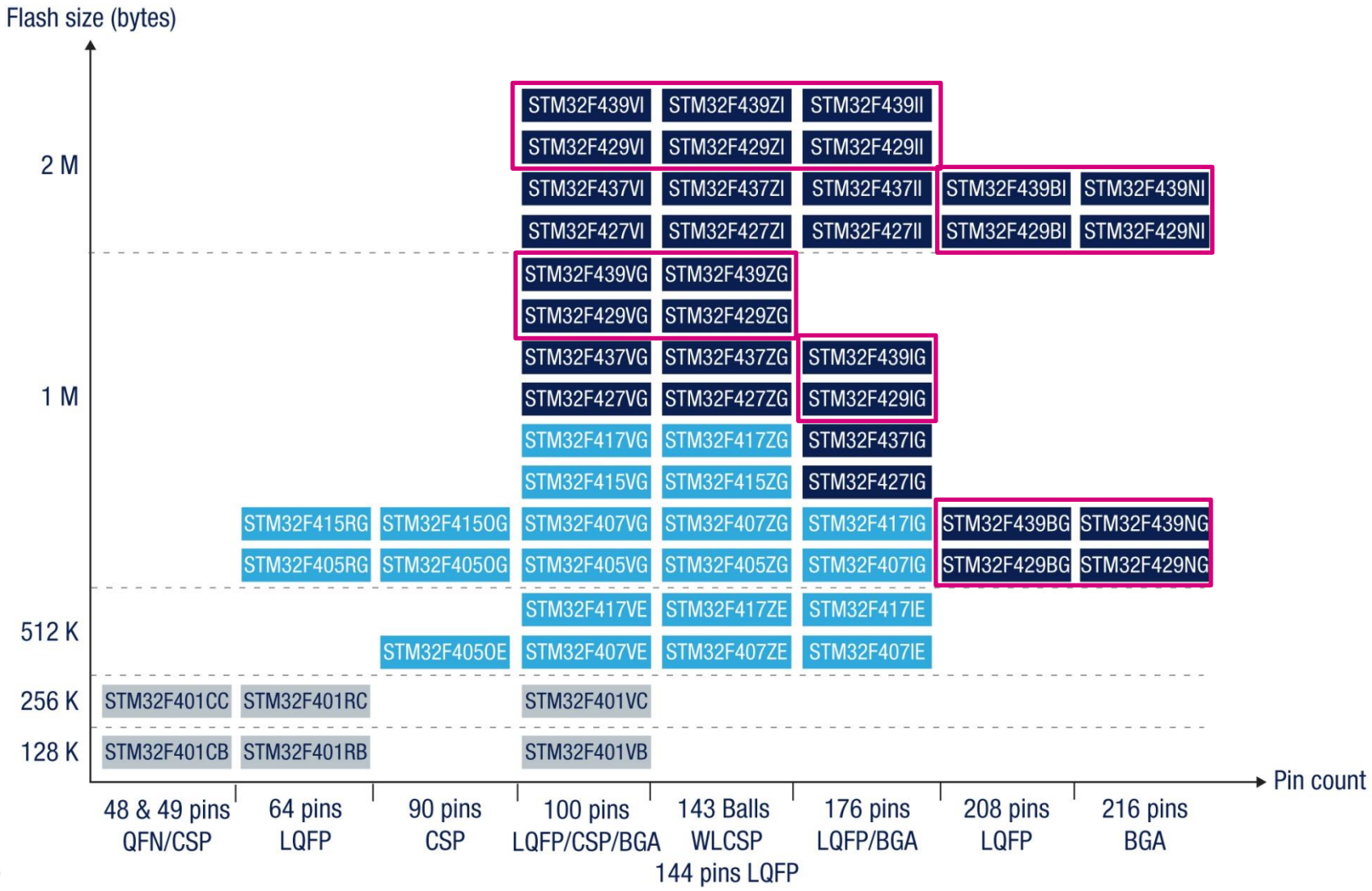
168 MHz 512-KB to 1-MB Flash 192-KB SRAM	Hardware Crypto /hash² RNG	2x 12-bit DAC

STM32F401

84 MHz 128-KB to 256-KB Flash 64-KB SRAM	• Power efficient:
	  • Run mode down to 140 µA/MHz
	  • Stop mode down to 11 µA typ
	• Small form factor: down to 3 x 3 mm

life.augmented

STM32 F4 portfolio



life.augmented

STM32F427/437/429/439

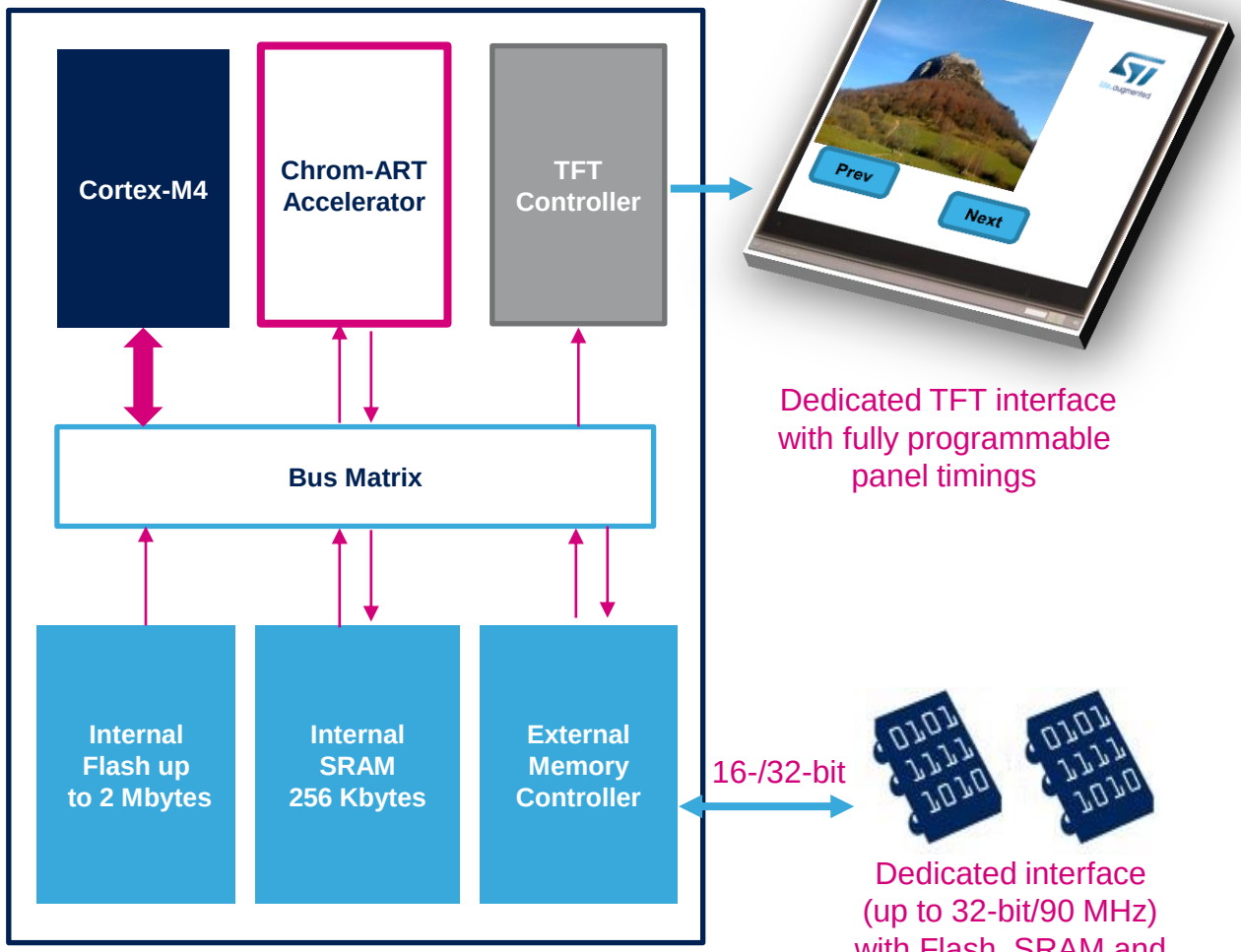
- Packages
  - WLSCP143 (<4.5x5.6mm)
  - LQFP100
  - LQFP144
  - LQFP176
  - BGA176
  - LQFP208
  - BGA216
- Operating voltage
  - 1.7 to 3.6V
- Temperature range
  - -40C to 85 °C
  - -40C to 105 °C

Notes:
1. HS requires an external PHY connected to the ULPI interface
2. Crypto/hash processor on STM32F415, STM32F417, STM32F437 and STM32F439
3. With digital filter feature, up to 1 Mbit/second
4. For STM32F4x9 only

STM32F429 as **HMI* controller**

STM32F4x9 using **Chrom-ART Accelerator offers twice more performance for**

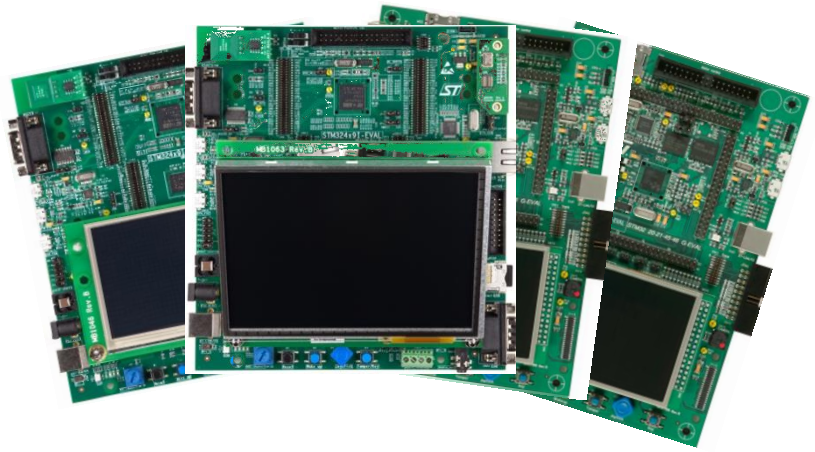
- Up to VGA/SVGA
- 16-/32-bit external memory interface
- Recommended packages: LQFP144,LQFP176/BGA176 or LQFP208/BGA216



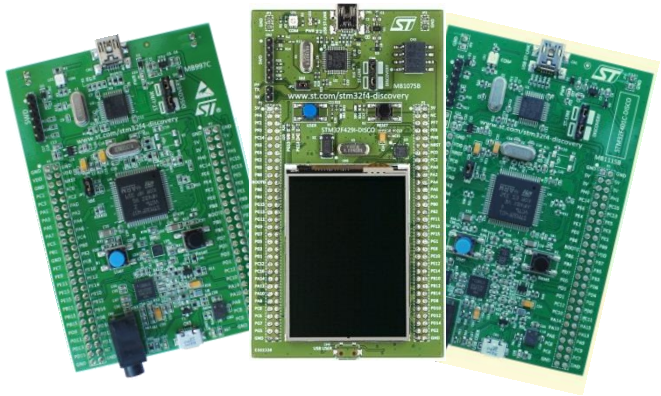
*HMI : Human Machine Interface

STM32F4 specific Hardware

- Evaluation boards:
  - Large offering of evaluation boards:
    - STM3240G-EVAL
    - STM3241G-EVAL
    - STM32429I-EVAL
    - STM32439I-EVAL



- Discovery kits:
  - STM32F4DISCOVERY
  - 32F401CDISCOVERY
  - 32F429IDISCOVERY

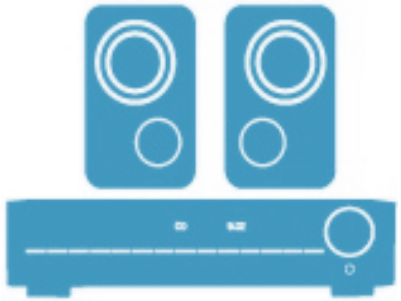


www.st.com/stm32f4-discovery

STM32F4 optimal software

- Graphical Stack -> www.st.com/stemwin
  - SEGGER and ST signed an agreement around emWin graphical stack. The solution is called **STemWin**:
    - Professional well-known stack solution
    - All emWin Widgets and PC Tools: GUIBuilder, simulator, widgets
    - Free for all STM32, delivered in binary
    - **Takes benefit from STM32F4 Chrom-ART Accelerator!**

- Audio offer: Full solution optimized for STM32F4
  - Full collection of codecs:
    - MP3, WMA, AAC-LC, HE-AACv1, HE-AACv2, Ogg Vorbis, G711, G726, IMA-ADPCM, Speex, …
  - ST Post Processing Algorithms:
    - Sample Rate Converters
    - Filters with examples like Bass Mix, Loudness….
    - Smart Volume Control: volume increase with no saturation
    - Stereo Widening

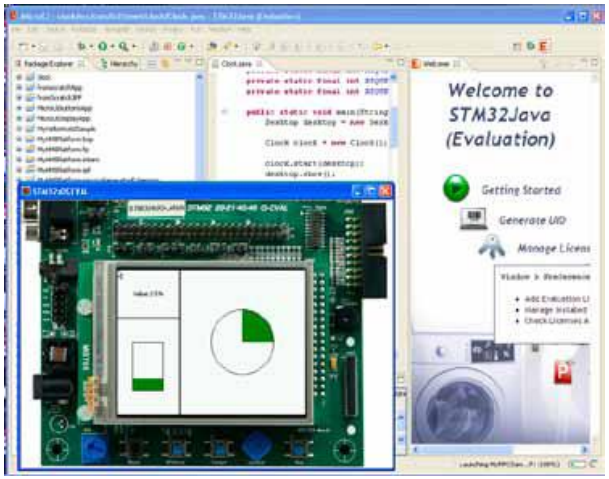


STM32F4 advanced Solutions

- **Beyond C Language !**

- Java evaluation kit:
  - Complete platform to evaluate the development of embedded applications in Java for the STM32 F4 series microcontrollers.

  -> www.stm32java.com

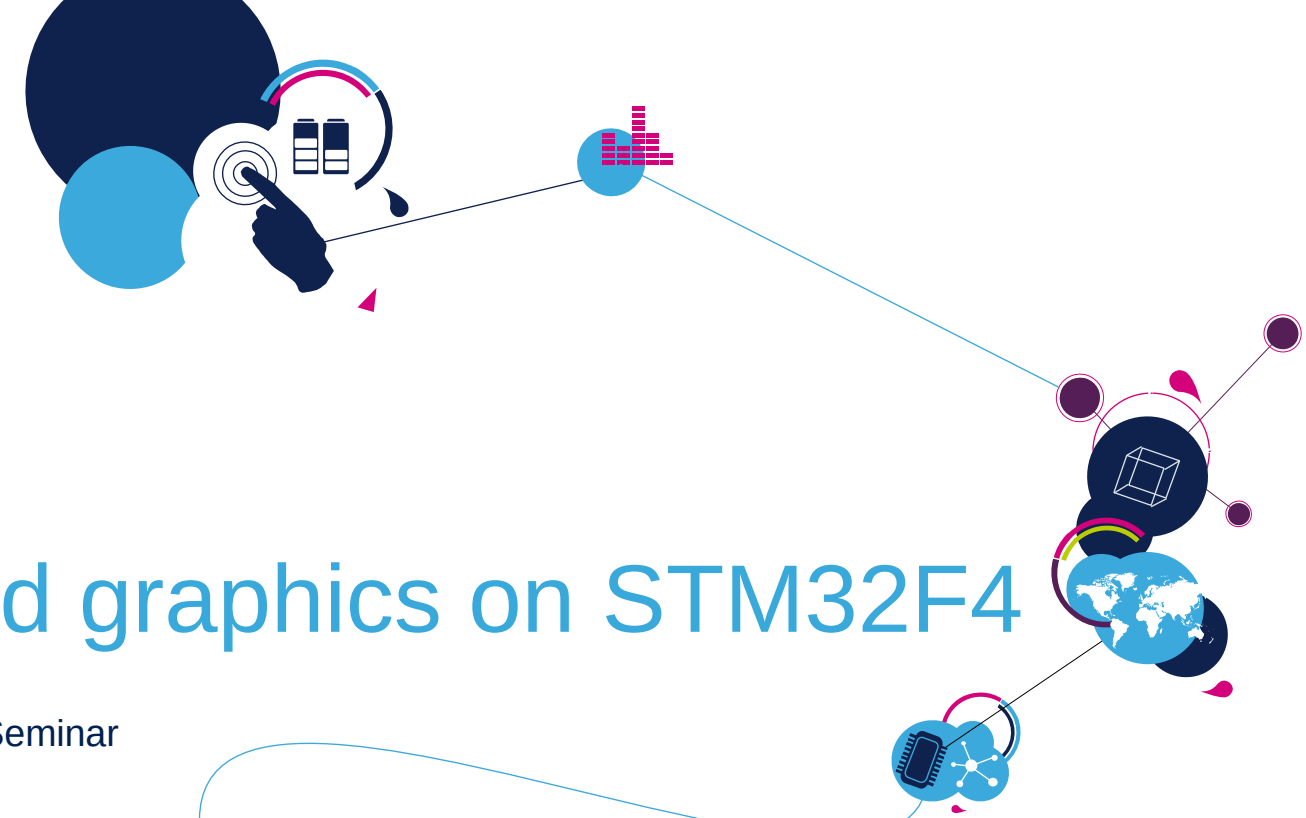


STM3240G-JAVA

- .Net Micro framework
  - Full support for Microsoft .Net Micro Framework
  - Full support for Microsoft Gadgeteer hobbyists initiative

  -> STM32F4-NETMF

STM3240G-ETH/NMF + STM3240G-USB/NMF



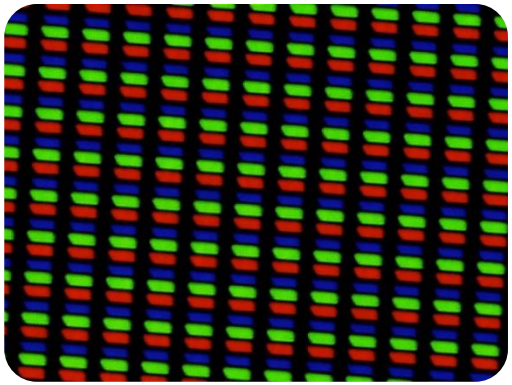
Embedded graphics on STM32F4

STM32F429-Discovery Seminar

life.augmented

Objectives

- Getting familiar with basics of graphics
  - General LCD connection
  - Color representation
  - Layers
  - Transparency / alpha channels
  - CLUT (Color Look Up Table)
  - Color keying



- Understand how you can benefit from STM32F4's HW acceleration
  - Usage of LTDC layer features (LCD-TFT Display Controller)
  - Offload CPU by using Chrom-ART
  - Hardware pixel format conversion

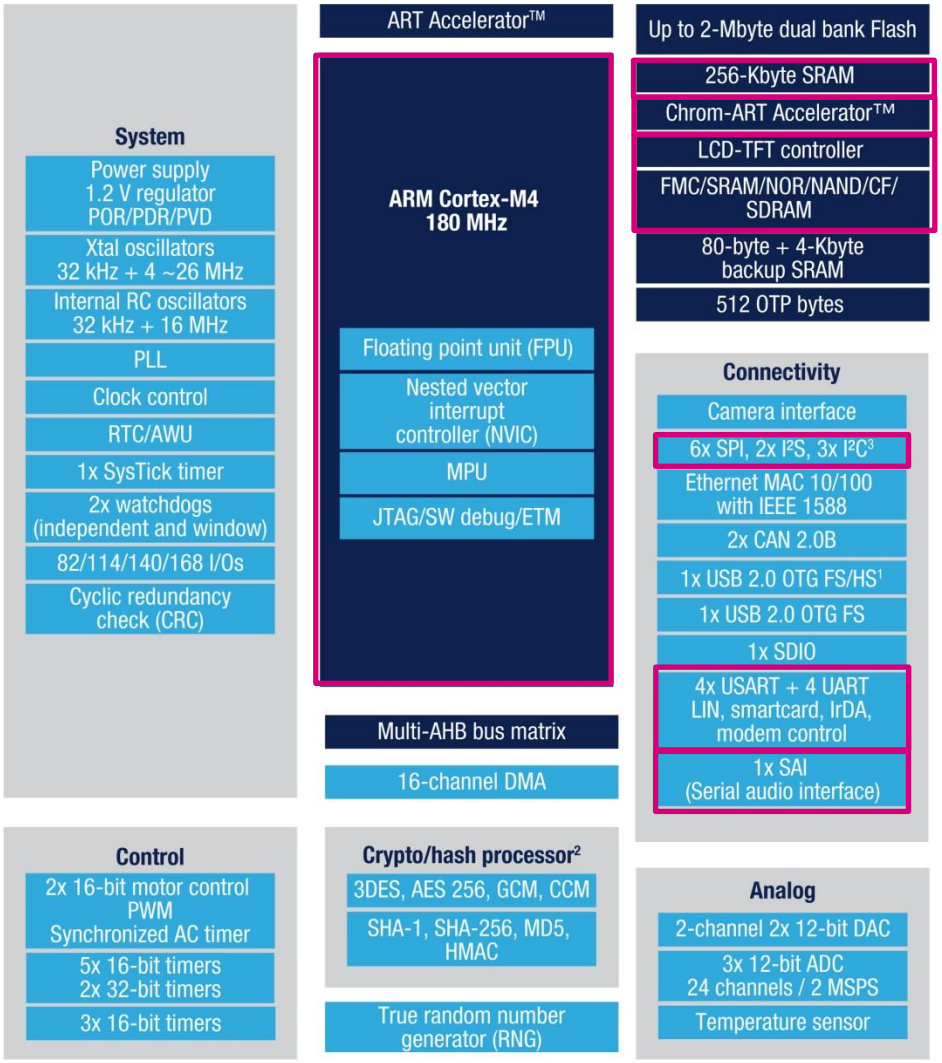
Contents

- Objectives
- STM32 internal architecture
- LCD-TFT controller (LTDC)
- Chrom-ART Accelerator (DMA2D)
- SDRAM interface

STM32F4x9 Block Diagram

- Fully compatible with F4x
- Up to 180MHz with overdrive mode
- Dual Bank 2 x 1MB Flash
- 256KB SRAM
- FMC with SDRAM + Support and 32-bit data
- Audio PLL + Serial Audio I/F
- LCD-TFT Controller
- Chrom – ART Accelerator
- Hash: supporting SHA-2 and GCM
- More serial comms and more fast timers running at FCPU
- 100 pins to 208 pins
- 1.71V - 3.6V Supply

New IPs/Features/more IPs instances



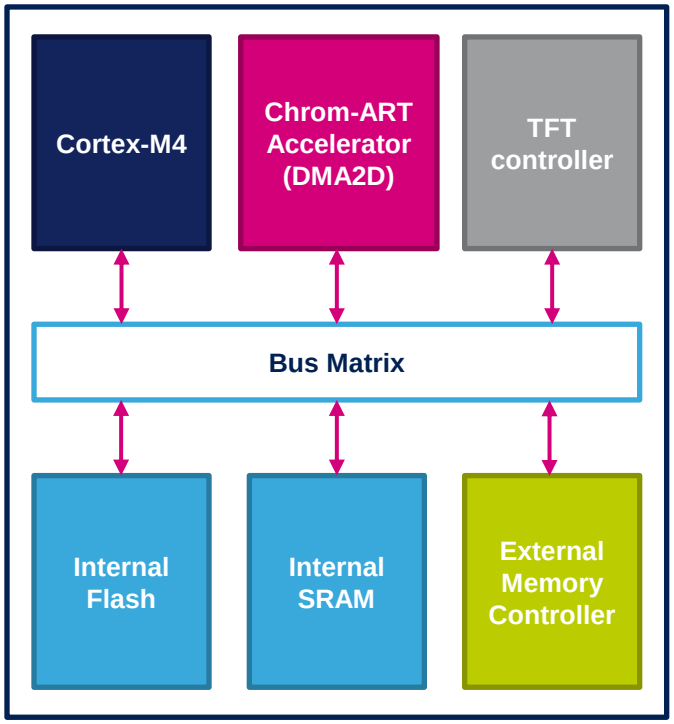
Notes:
1. HS requires an external PHY connected to the ULPI interface
2. Crypto/hash processor on STM32F439 only
3. With digital filter feature, up to 1 Mbit/second

life.augmented

Graphics on STM32

life.augmented

STM32F42x9 Architecture



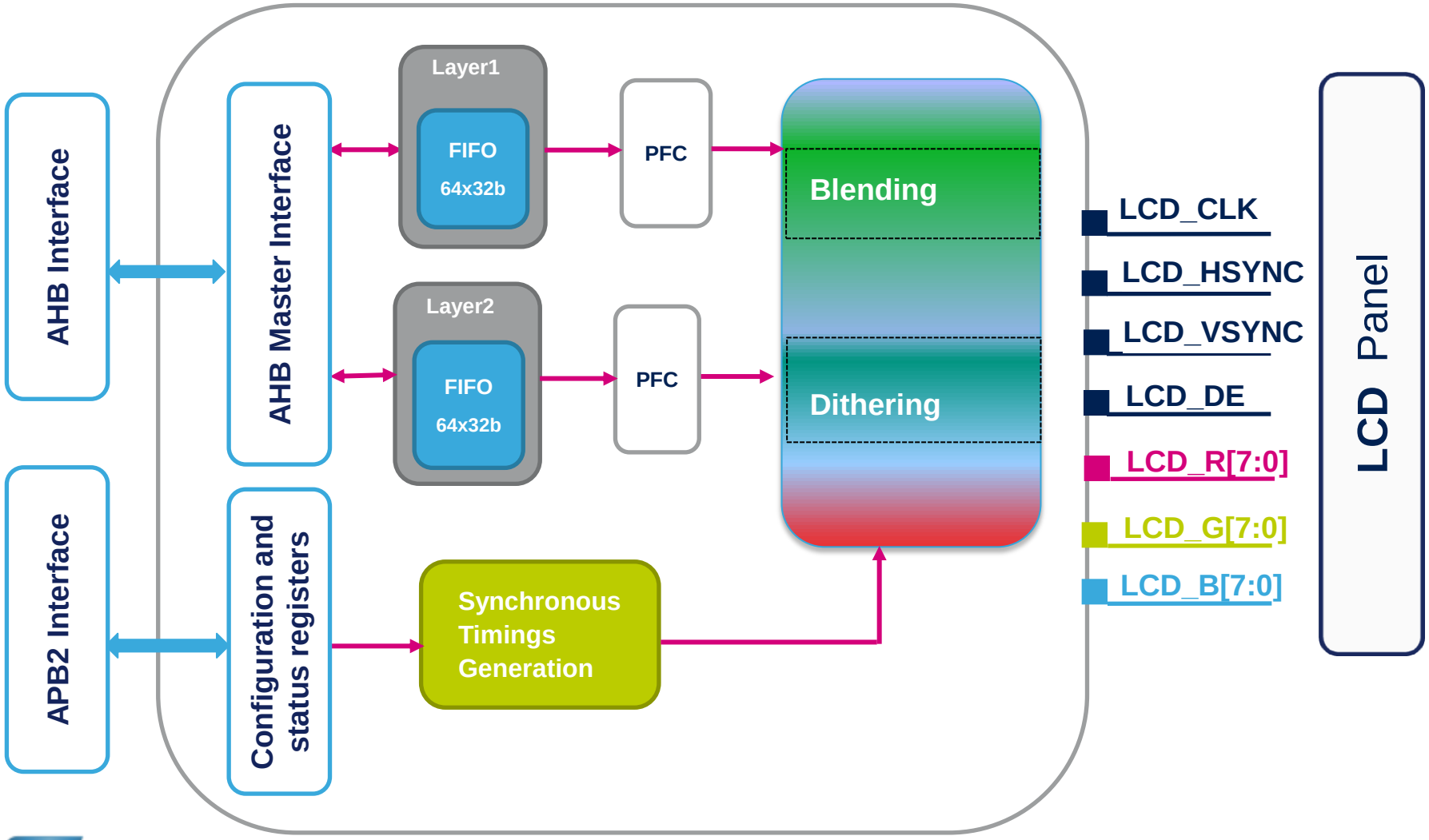
External memory

- **TFT controller** allows the interfacing
- **Chrome-ART (**DMA2D) provides a **true HW acceleration**
- **DMA2D offloads the CPU** for operations like rectangle **graphic** filling, rectangle copy (with or without pixel format conversion), and image blending
- **DMA2D goes faster than the CPU** for the equivalent operation

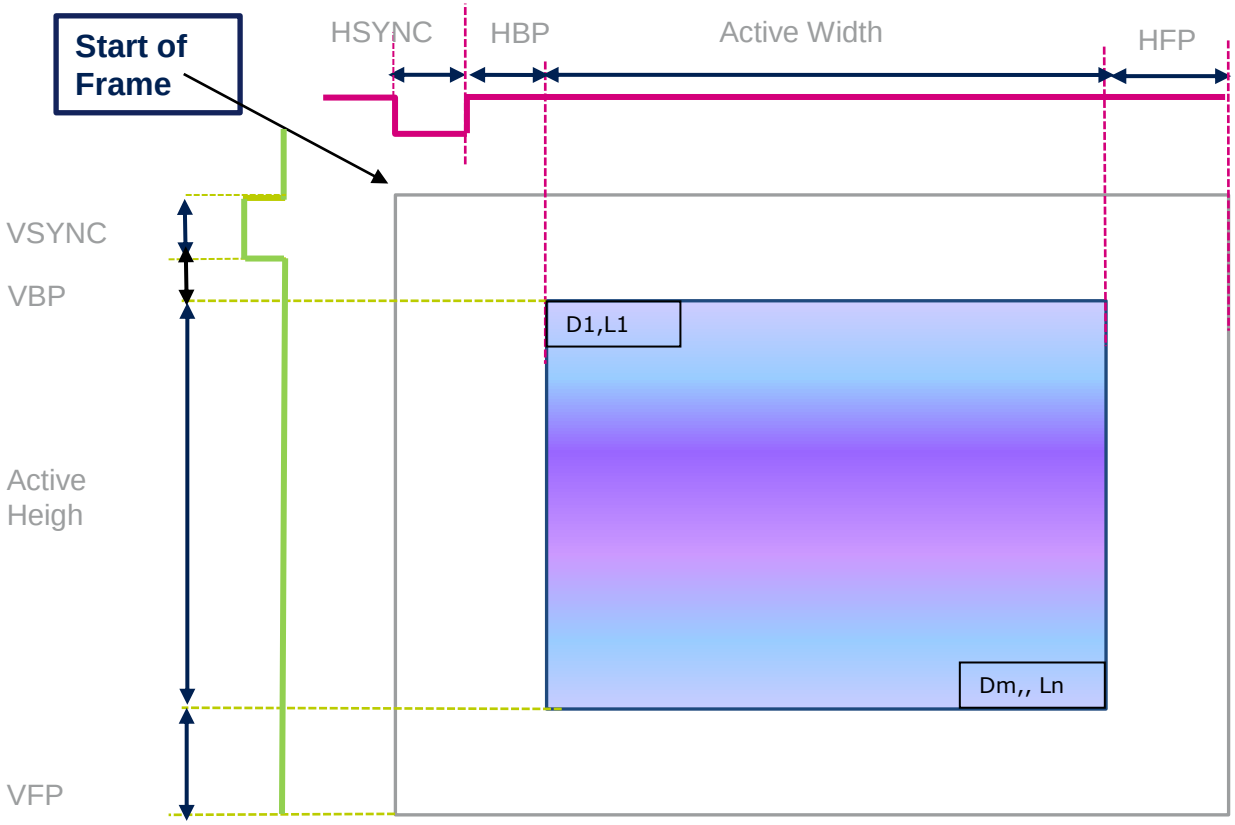
LCD-TFT Display Controller (LTDC)

life.augmented

LCD-TFT architecture



LTDC Timings



VBP: Vertical Back porch

VFP: Vertical Front porch

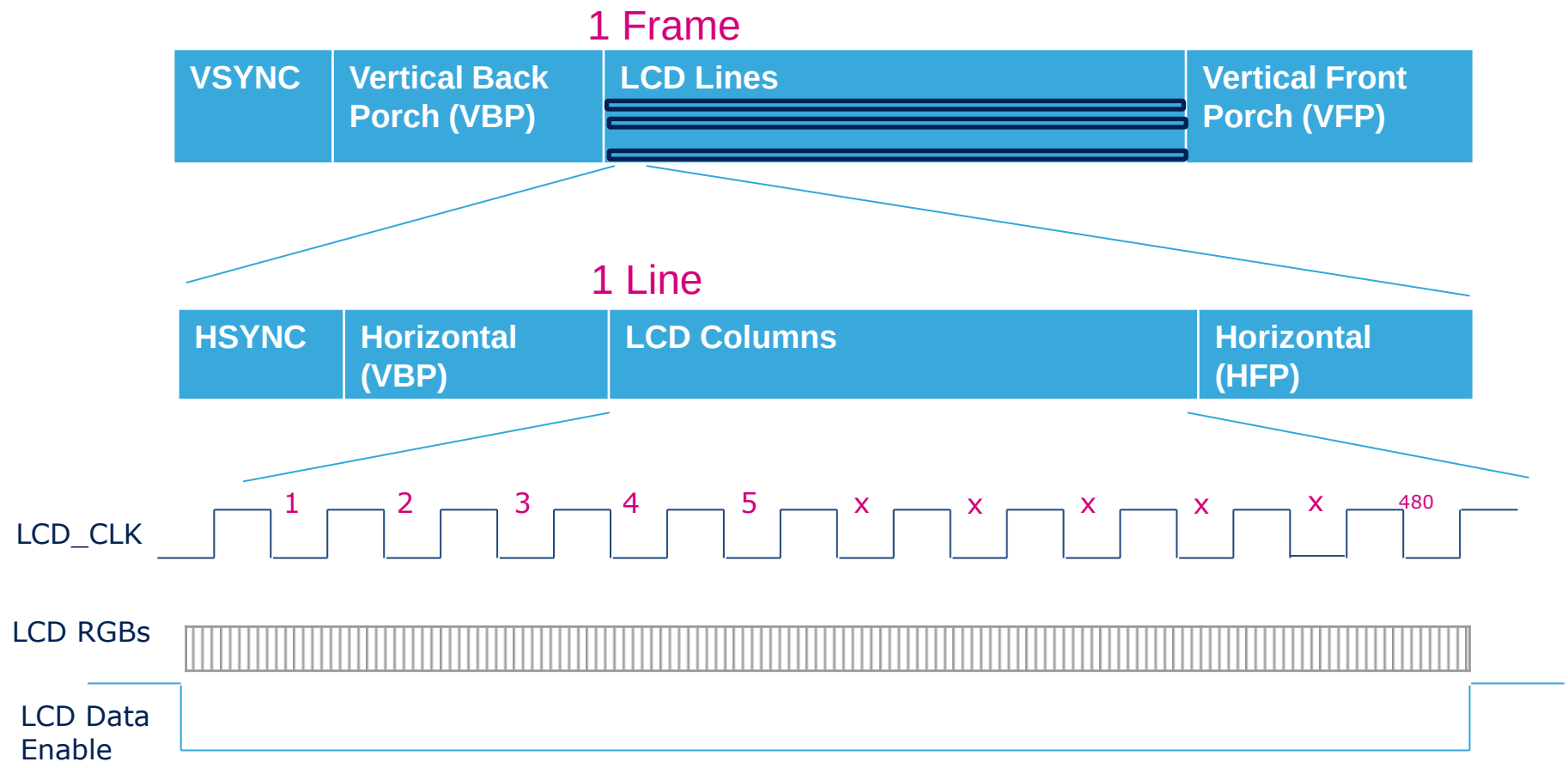
HBP: Horizontal Back porch

HFP: Horizontal Front porch

These timings come from the datasheet of the display. E.g. TFT on discovery kit.

Parameters	Symbols	Condition	Min.	Typ.	Max.	Units
Horizontal Synchronization	Hsync		2	10	16	DOTCLK
Horizontal Back Porch	HBP		2	20	24	DOTCLK
Horizontal Address	HAdr		-	240	-	DOTCLK
Horizontal Front Porch	HFP		2	10	16	DOTCLK
Vertical Synchronization	Vsync		1	2	4	Line
Vertical Back Porch	VBP		1	2	-	Line
Vertical Address	VAdr		-	320	-	Line
Vertical Front Porch	VFP		3	4	-	Line

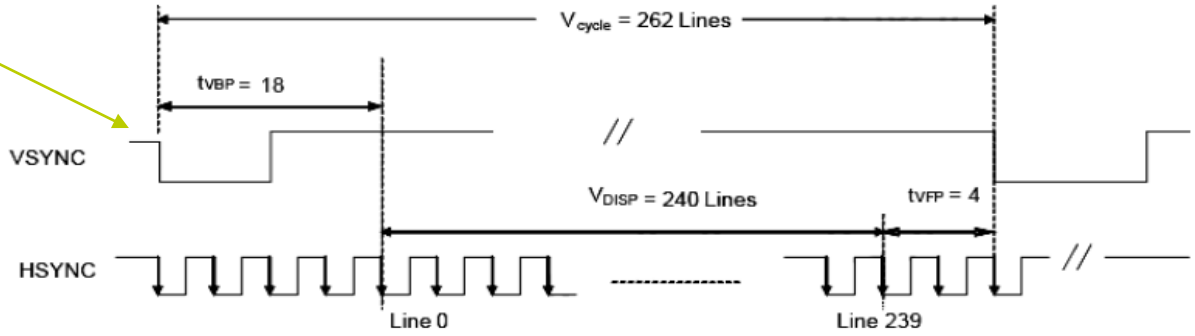
Frame Display



Vertical/Horizontal transactions

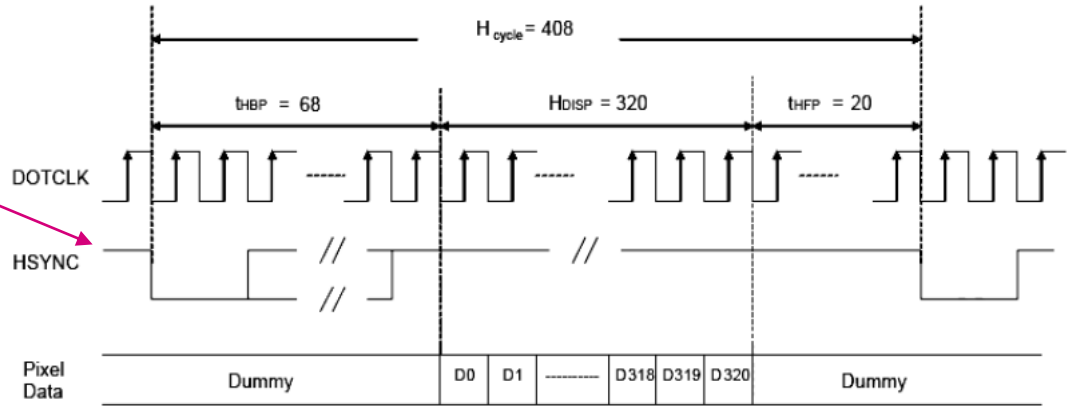
- Vertical transaction

Start of Frame



- Horizontal transaction

Start of Scan Line



life.augmented

LCD basic timing signals

- LCDs require the following basic timing signals:
  - VSYNC (Vertical Sync for TFT)
    - **Used to reset the LCD** row pointer to the top of the display
  - HSYNC (Horizontal sync for TFT)
    - Used **to reset the LCD column** pointer to the edge of the display
  - D0..Dxx (1 or more data lines)
    - Data line
  - LCDCLK (LCD pixel clock)
    - Used to panel control refresh rate
- Some panels may require additional timing signals:
  - LCD_DE
    - LCD data Enable. Used to indicate valid data on the LCD data bus
  - Other signals –some optional
    - LCD power, backlight power, touchscreen

LCD Memory Requirements

- **Frame buffer size**
  - The panel size and bits per pixel determine the amount of memory needed to hold the graphic buffer.
  - **Memory Requirement (KiloBytes) = (bpp * Width * Height) / 8**
  - In many cases, more memory is needed. E.g. **double buffering**: one graphic buffer to store the current image while a second buffer used to prepare the next image.

Panel Resolution	Total Pixel	bpp (Bit per pixel)	Required memory (KB)
320x240 (QVGA)	76.8K	16bpp	153.6
		8bpp	76.8
480x272 (WQVGA)	130.5K	16bpp	216.1
640x480 (VGA)	307.2K	16bpp	614.4
800x600 (SVGA)	480K	16bpp	960

LTDC Clocks and reset

- **Three** clock domains:
  - AHB clock domain (HCLK)
    - **To transfer da**ta from the memories to the Layer FIFO and vice versa
  - APB2 clock domain (PCLK2)
    - **To access** the configuration and status **registers**
  - The Pixel Clock domain (LCD_CLK)
    - **To generate** LCD-TFT interface signals.
    - LCD_CLK output should be configured following the panel requirements. The LCD_CLK is configured through the PLLSAI
- LTDC Reset
  - It is reset by setting the LTDCRST bit in the RCC_APB2RSTR register

LCD-TFT Signals

- The LCD-TFT I/O pins not used by the application can be used for other purposes.

LCD-TFT Signals	Description
LCD_CLK	Pixel Clock output
LCD_HSYNC	Horizontal Synchronization
LCD_VSYNC	Vertical Synchronization
LCD_DE	Data Enable
LCD_R[7:0]	8-Bits Red data
LCD_G[7:0]	8-Bits Green data
LCD_B[7:0]	8-Bits Blue data

LTDC Main Features - (1/3)

- 24-bit RGB Parallel Pixel Output; 8 bits-per-pixel (RGB888)
- AHB 32-Bit master with burst access of 16 words to any system memory
  - Dedicated FIFO per Layer (depth of 64 words)
- Programmable timings to adapt to targeted display panel.
  - HSYNC width, VSYNC width, VBP, HBP, VFP, HFP
- Programmable Polarity of:
  - HSYNC, VSYNC, Data Enable
  - Pixel clock
- Supports only TFT (no STN)

LTDC Main Features - (2/3)

- Programmable display Size
  - Supports up to **800 x 600 (SVGA)**
- Programmable Background color
  - 24-bit RGB, used for blending with bottom layer
- Multi-Layer Support with blending, **2 layers + solid color background**
- Dithering (2-bits per color channel (2,2,2 for RGB))
  - Combination of adjacent pixels to simulate the desired shade
  - Dithering is applicable for 18bit color displays, to simulate 24bit colors.
- New programmed values can be loaded immediately at run time or during Vertical blanking
- 2 Interrupts generated on 4 event Flags

Alpha Channel Usage

- The Alpha channel represents the transparency of a pixel
- It's **mandatory** as soon as start manipulating bitmaps with non square edges or for anti-aliased font support

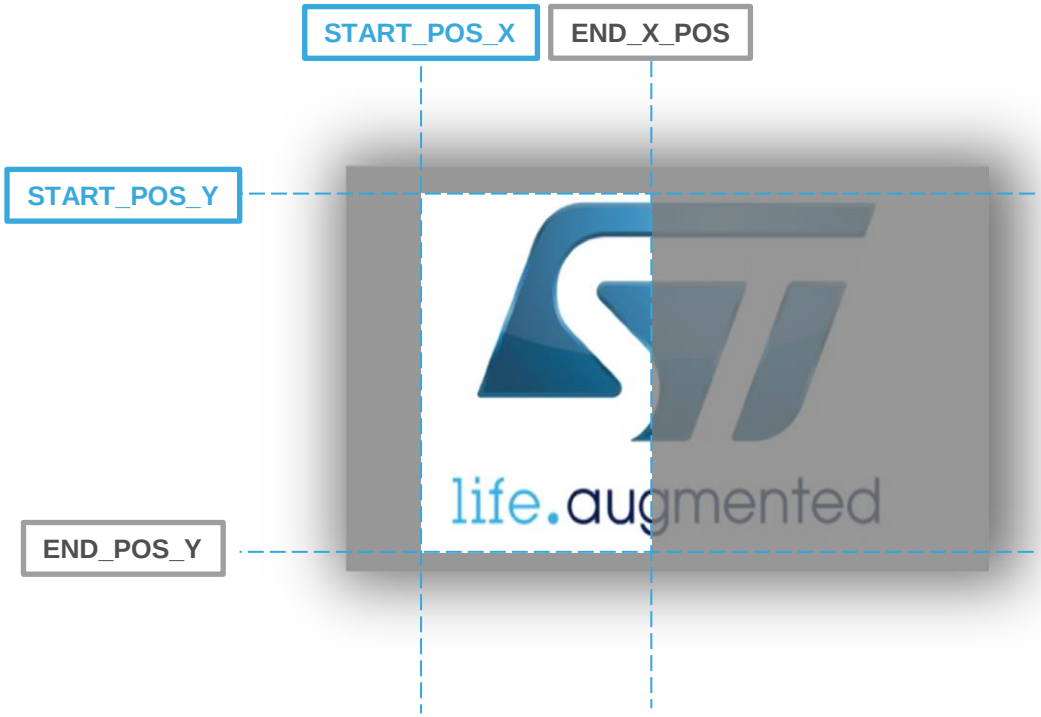
Aliased

Anti-aliased

- **0xFF = opaque**
- 0x00 = transparent

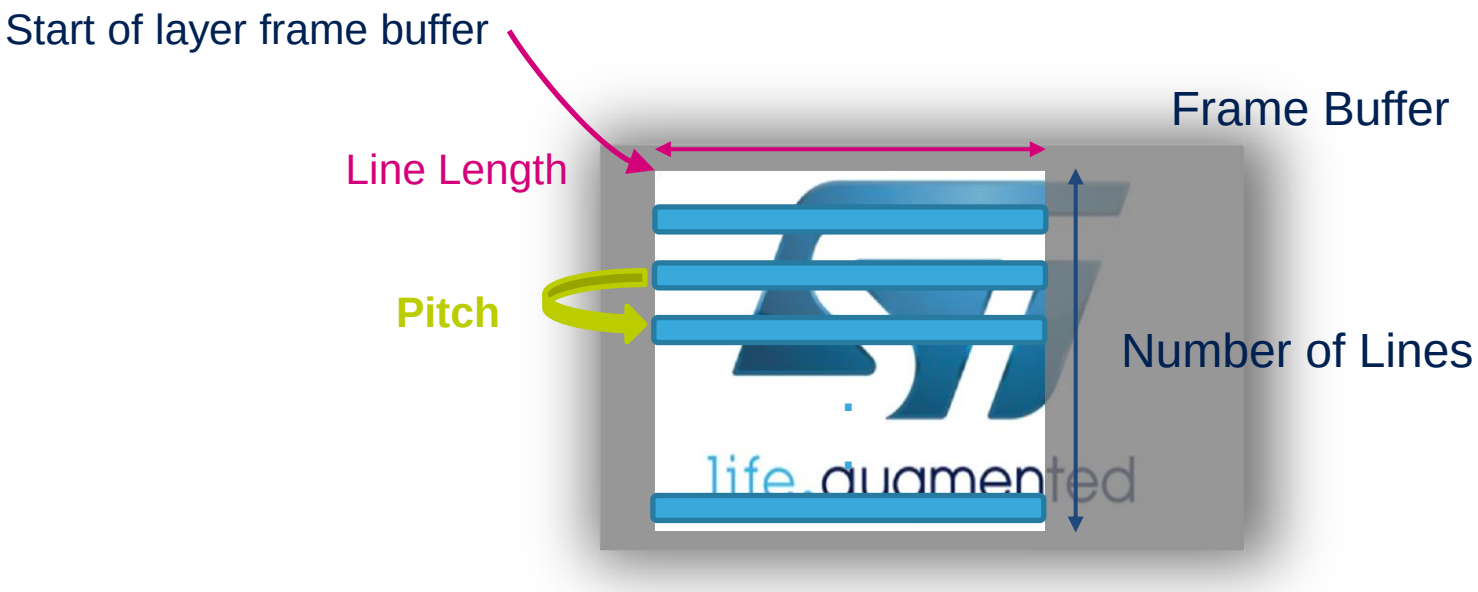
Layer Programmable Parameters: Window

- Window **position** and **size**
  - The first and last visible Pixel are configured in the **LTDC_LxWHPCR** register.
  - The first and last visible line are configured in the **LTDC_LxWVPCR**



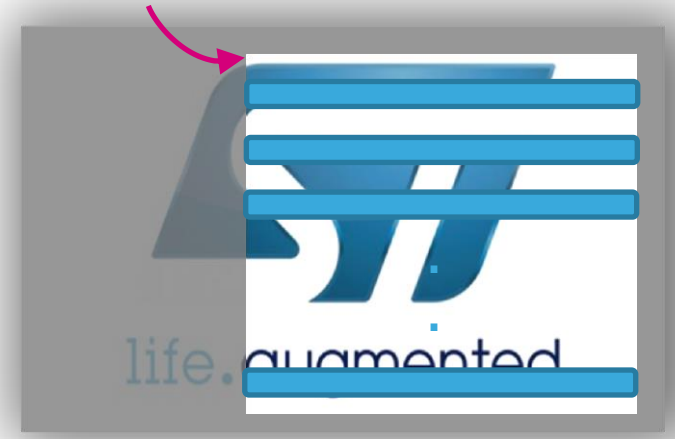
Layer Programmable Parameters: Color Frame Buffer

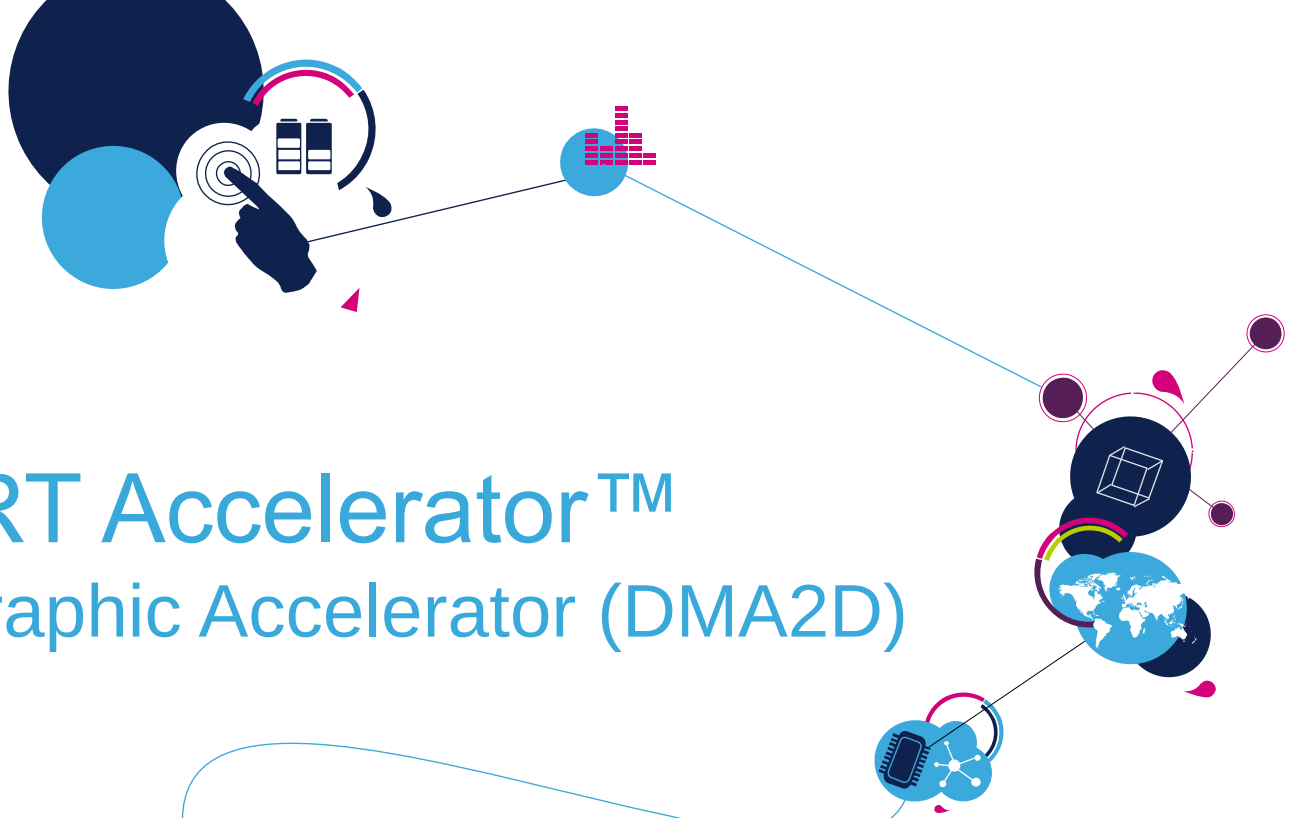
- The frame buffer size, line length and the number of lines settings must be correctly configured :
  - If it is set to **fewer bytes** than required by the layer, a **FIFO under run** error will be set.
  - If it is set to **more bytes** than actually required by the layer, the useless data read is discarded. The useless data is **not displayed**.



Layer Programmable Parameters: Color Frame Buffer - effects

- Simply changing the start of the layer frame buffer you can scroll the picture over the layer
  - The picture stored in the memory must be bigger than actual layer size





Chrom-ART Accelerator™

STM32F4 Graphic Accelerator (DMA2D)

life.augmented

Graphic content creation

Creating something « cool »

- How the frame buffer is generated for creating a “cool” graphical interface to be displayed through the TFT controller ?

Frame buffer construction

- The frame buffer is generated **drawing successively** all the graphic objects

Frame buffer generation needs

- The resulting frame buffer is an **uncompressed bitmap** of the size of the screen with a color coding corresponding to the targetted display

  - Each object to be drawn can be
    - A **Bitmap** with its own color coding (different from the final one), compressed or not
    - A **Vectorial Description** (a line, a circle with a texture...etc....)
    - (A **Text**)

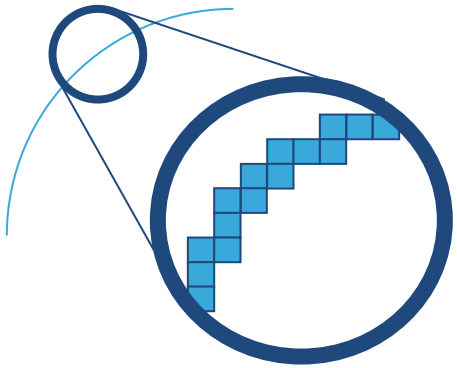
Bitmap or Vector

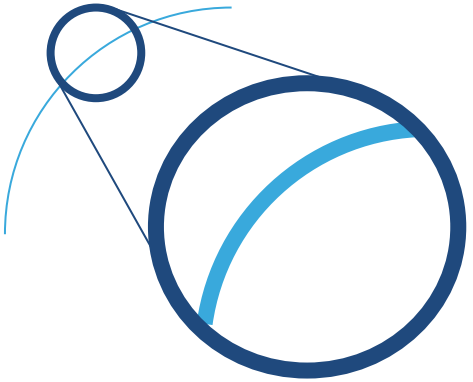
Bitmap

- High ROM usage
- No CPU usage if no compression, but can be needed for uncompression
- Geometrical transformations limited and would need filtering
- Description standard bitmap files that are converted into C table

Vector graphics

- Low ROM usage
- High CPU usage as image needs to be generated : **a GPU is mandatory**
- Geometrical transformations are natural
- Description through
  - Polygons : 3D graphics
  - Curves : 2D graphics

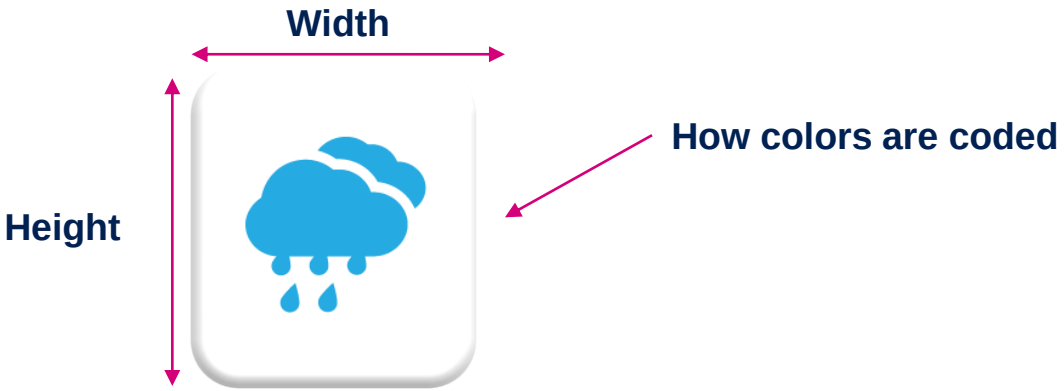




Bitmaps

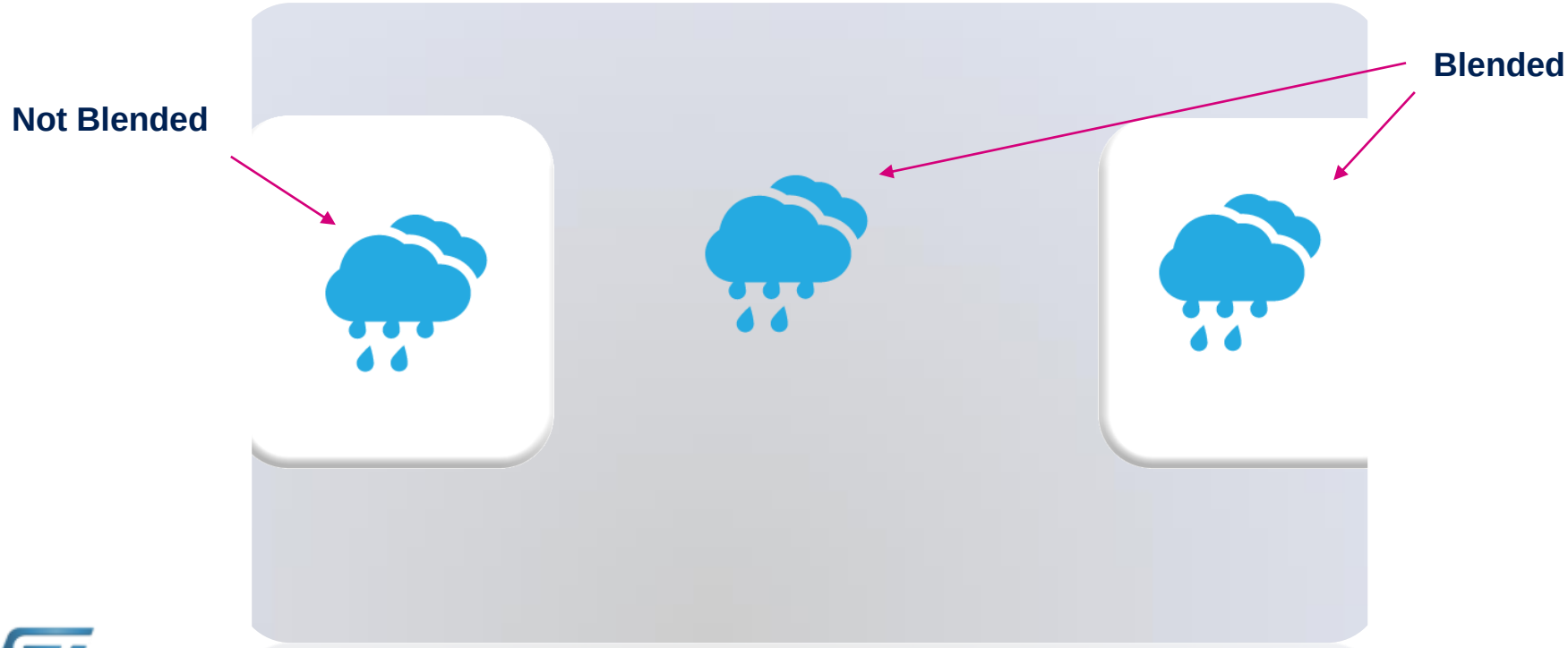
Overview

- Bitmaps are an array representation of an image
- It shall have the **at least** following properties
  - **Width** (in pixel)
  - **Height** (in pixel)
  - **Color mode** (direct, indirect, ARGB8888, RGB565...etc...)
  - Color Look Up Table (optional)



Blending

- Blending consists of drawing an image onto another while respecting the transparency information
- As a consequence, blending implies reading 2 sources, then blending, then writing to the destination



Back to our « cool » interface

Background

Almost Uniform
L8 mode

Button

Round shape
Gradient
ARGB8888 mode

Icon

Complex shape
Many colors
ARGB8888 mode

Fonts

Specific management with A8 or A4 mode

Font management

- Bitmap fonts are managed only using alpha channel (transparency)

ROMed character bitmap (A8 or A4)

PFC

Generated character bitmap with color information (ARGB8888, ARGB1555, ARGB4444) or (RGB888, RGB565)

Summary

- To generate the 24bpp frame buffer:
  - Copy background from the ROM to the frame buffer with PFC L8 to RGB888
  - Copy buttons & icons from the ROM to the frame buffer with blending
  - Copy characters from the ROM to the frame buffer with PFC A4 to ARGB8888 and blending

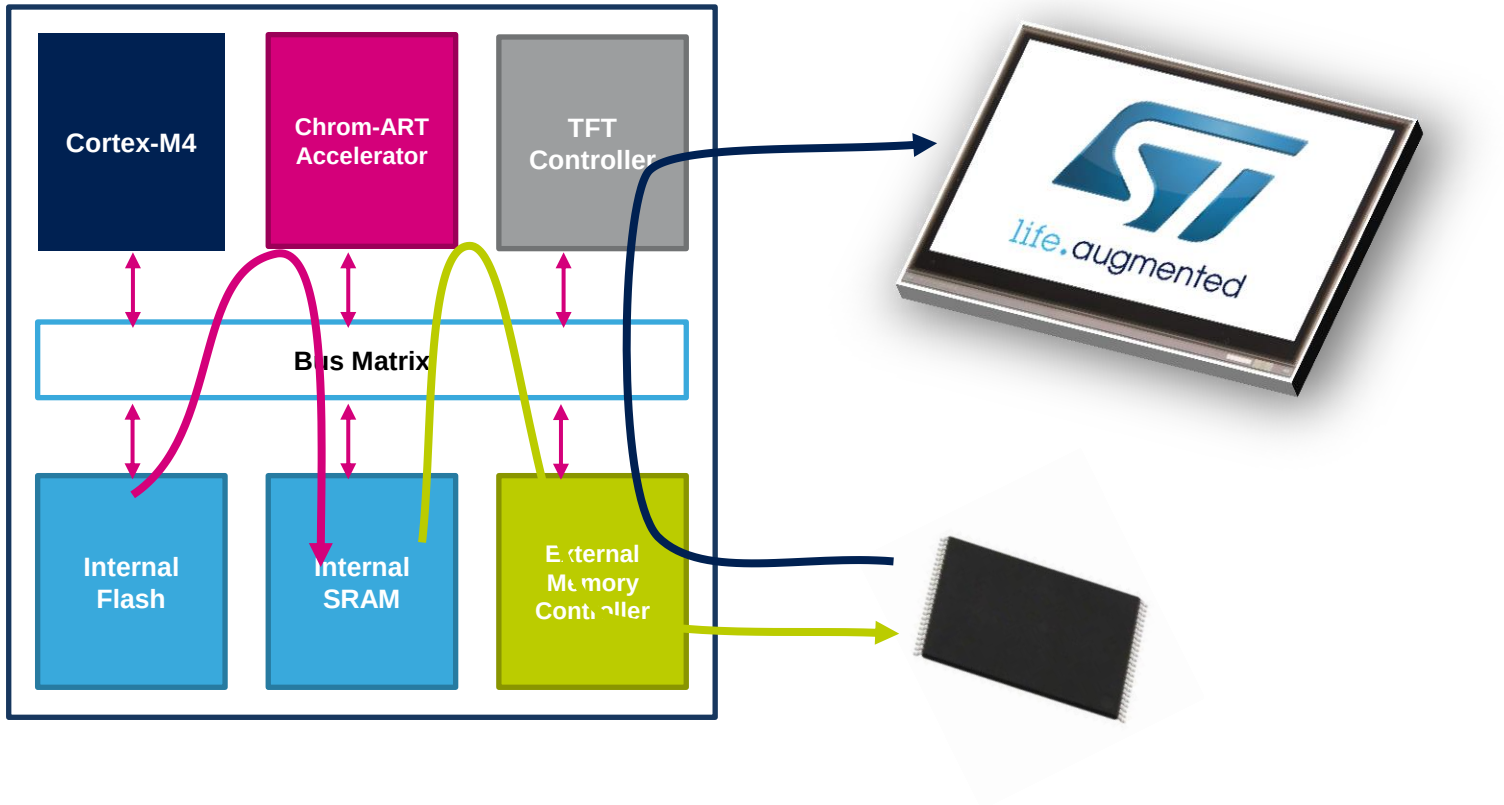
- Many copy operations with pixel conversions can be done by the CPU, but it's very time consuming → HW acceleration helps.

Chrom-ART Accelerator (DMA2D)

Overview

- The Chrom-ART combines both a DMA2D and graphic oriented functionality for image blending and pixel format conversion.
- To **offload the CPU** of raw data copy, the Chrom-ART is able to copy a part of a graphic into another part of a graphic, or simply fill a part of a graphic with a specified color.
- In addition to raw data copy, additional functionality can be added such as image format conversion or image blending (**image mixing with some transparency**).

Typical data flow



Creation of an object in a memory device by the Chrom-ART

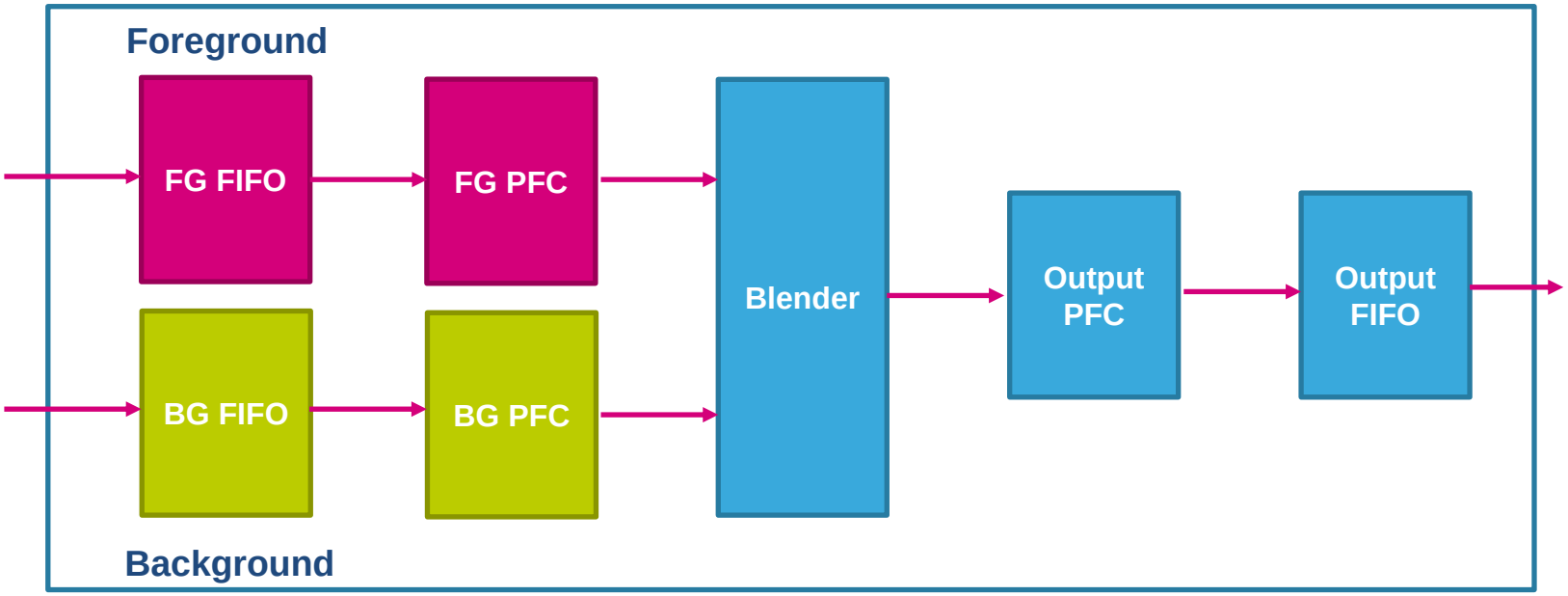
Update/Creation of the frame buffer in the external RAM by the Chrom-ART

TFT controller data flow

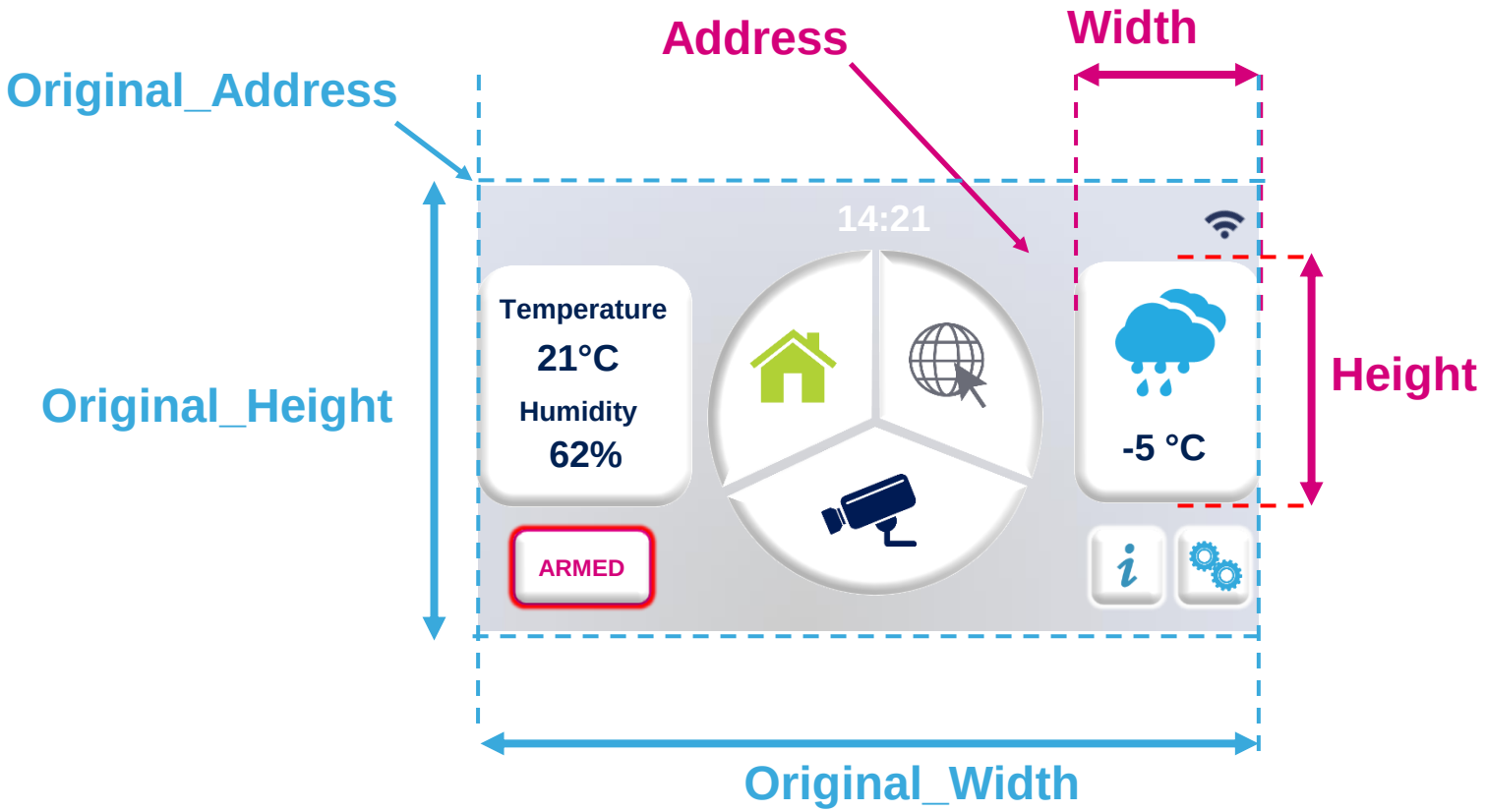
Chrom-ART features

- AHB bus master with burst access to any system memory
- Programmable bitmap **height, width and address** in the memory
- Programmable data format (from 4-bit indirect up to 32-bit direct)
- **Dedicated memory** for color lookup table (CLUT) independent from LTDC
- Programmable address destination and format
- Optional image format conversion from direct or indirect color mode to direct color mode
- Optional blending machine with programmable transparency factor and/or with native transparency channel between to independent image input to the destination image.

Chrom-ART pixel pipeline



Bitmap parameter



Address = Original_Address + (X + Original_Width * Y) * BPP
LineOffset = Original_Width - Width

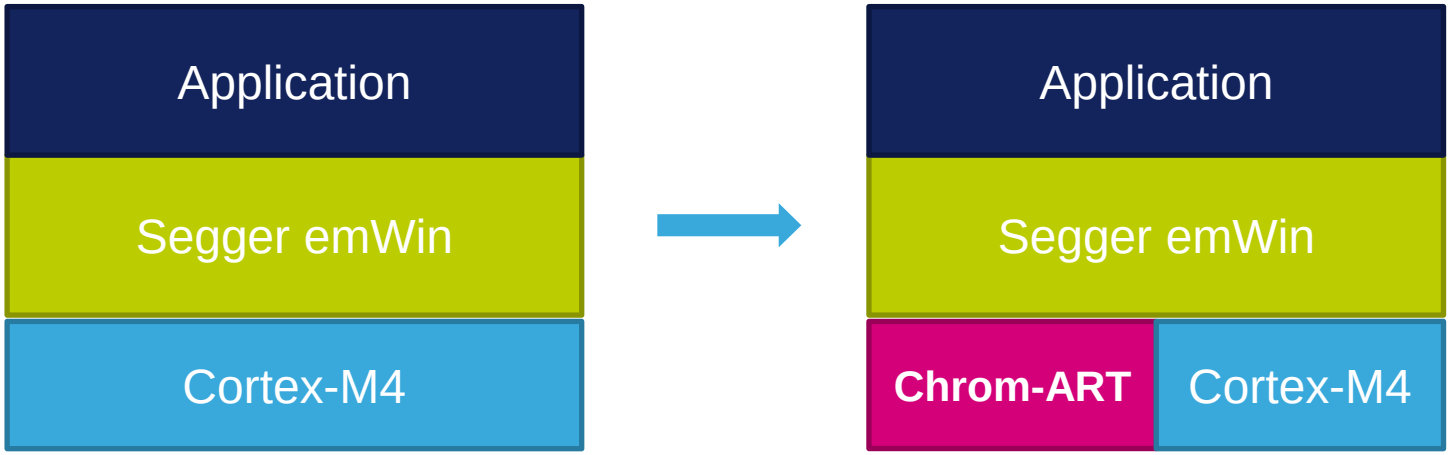
Chrom-ART performances

Rectangle Copy with Pixel Format Conversion

Source Format	Destination Format	Size (Word)	Gain (compared to CPU)
ARGB8888	RGB565	128	97.462685%
ARGB8888	ARGB4444	128	98.813454%
ARGB8888	ARGB1555	128	98.786247%
ARGB8888	RGB888	192	98.015434%
ARGB4444	RGB565	128	97.985779%
ARGB4444	ARGB8888	128	97.914253%
ARGB4444	ARGB1555	128	98.162163%
ARGB4444	RGB888	128	98.229164%
ARGB1555	RGB565	128	97.029701%
ARGB1555	ARGB4444	128	97.861633%
ARGB1555	ARGB8888	128	98.454933%
ARGB1555	RGB888	128	98.527000%
RGB565	ARGB8888	128	98.094170%
RGB565	ARGB4444	128	96.653542%
RGB565	ARGB1555	128	94.352165%
RGB565	RGB888	128	98.621246%
RGB888	RGB565	128	96.964287%
RGB888	ARGB4444	128	98.006645%
RGB888	ARGB1555	128	98.083069%
RGB888	ARGB8888	256	98.210526%

All measurements done on FPGA - Comparison with the CPU doing the same operation

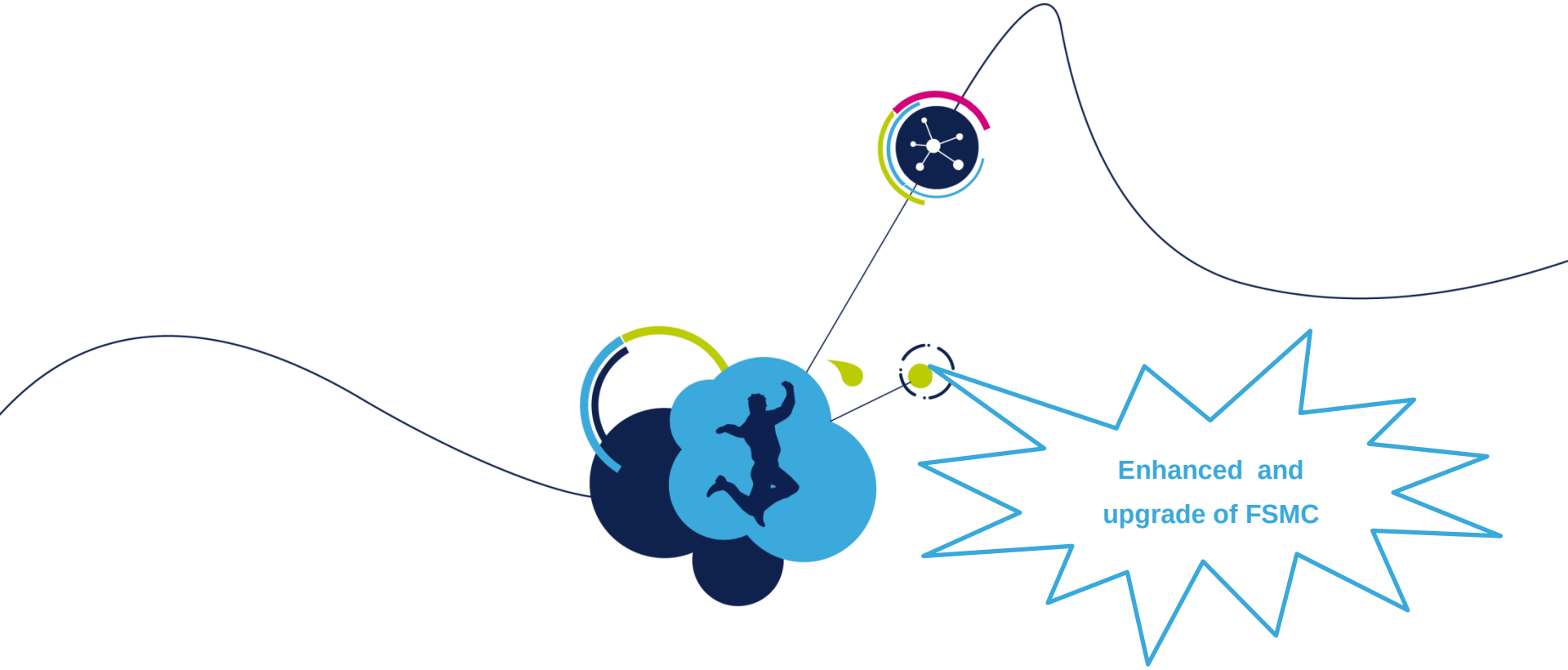
Integration with Graphic Library



- **Chrom-ART** integration is **transparent for the application**
- The low-level drivers of the graphical stack are upgraded to directly use **Chrom-ART** for data transfer, pixel format conversion and blending
- **CPU load is decreased** and **graphical operations are faster**

STM32 Solution for Graphics

- STM32F429 offers a unique graphic capability in the Cortex-M4 based MCU family
  - **Real TFT Controller** for optimum display control
  - External memory interface to connect both Flash for static content and SRAM or SDRAM for dynamic content and frame buffer
  - On-chip hardware acceleration deeply coupled with graphical library
  - Standard graphical library taking advantage of on-chip graphical acceleration



Flexible Memory Controller (FMC)

life.augmented

Objectives

- Overview of FMC interface
- Usage of high capacity RAM storage - SDRAM
  - For frame buffer
  - For picture storage and preparation
  - For animation

FMC Features (1/2)

- 6 Banks to support External memories
- FMC external access frequency is up **90MHz** when HCLK is at **180MHz, or 84 MHz when HCLK=168 MHz**
- Independent chip select control for each memory bank
- Independent configuration for each memory bank
- Programmable timings to support a wide range of devices
- 8/16/**32 bits** data bus
- External asynchronous wait control
- Interfaces with **Synchronous DRAM (SDRAM)** memory-mapped

FMC Features (2/2)

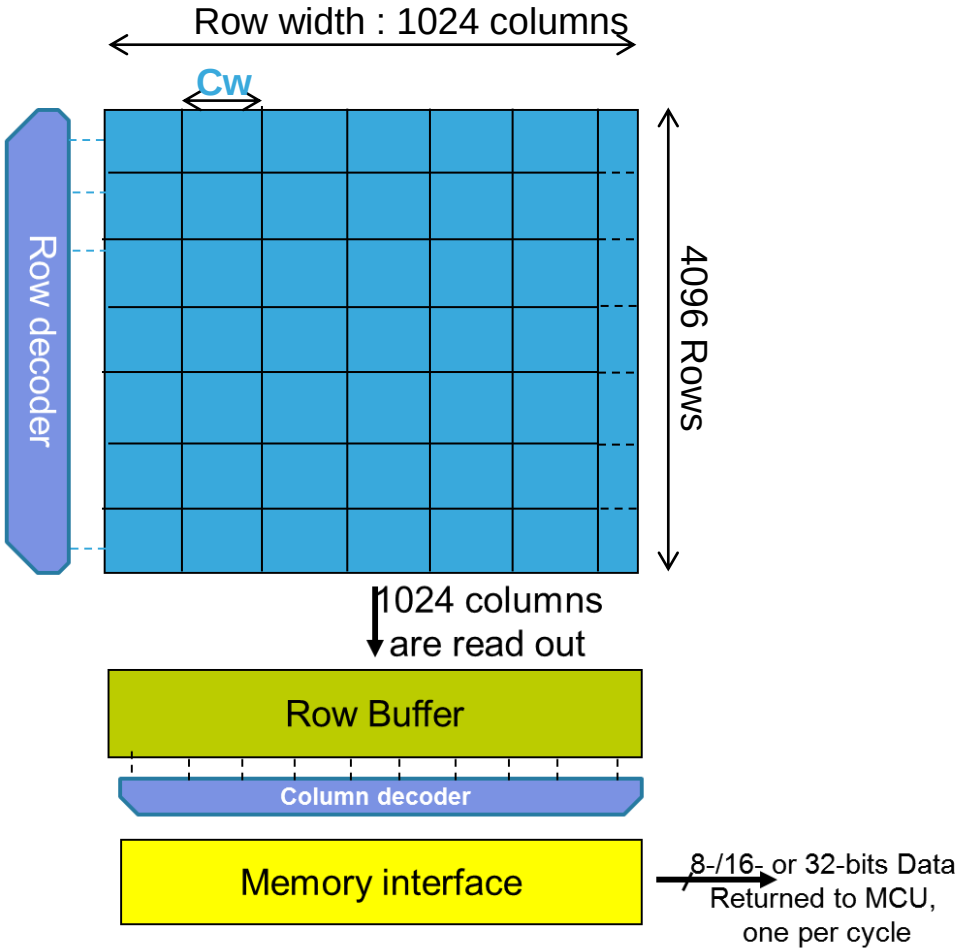
- Interfaces with static memory-mapped devices including:
  - static random access memory (SRAM)
  - read-only memory (ROM)
  - NOR/ OneNAND Flash memory
  - PSRAM
- Interfaces with parallel LCD modules: Intel 8080 and Motorola 6800
- Interfaces with NAND Flash and 16-bit PC Cards
  - With ECC hardware up to 8 Kbytes for NAND memory
  - 3 possible interrupt sources (Level, Rising edge and falling edge)
- Supports burst mode access to synchronous devices (NOR Flash and PSRAM)

SDR SDRAM protocol overview

life.augmented

SDRAM memory organization

- Example of DRAM array with 12-bits row and 10-bits column
- A row width is 2exp(10) = 1024 column
- Column width is 8-bit, 16-bit or 32-bit (Cw)



Memory latency computation

- Assuming an SDRAM with one internal bank memory and CAS Latency = 2, TRCD = 2, TRP = 2.
- Access order is A0, A2, B3, A1
  - Ax : with A is row index, x is column index
  - Bx : with B is row index, x is column index
- How many cycles it will require to read the 4x data?
- Three cases are observed :

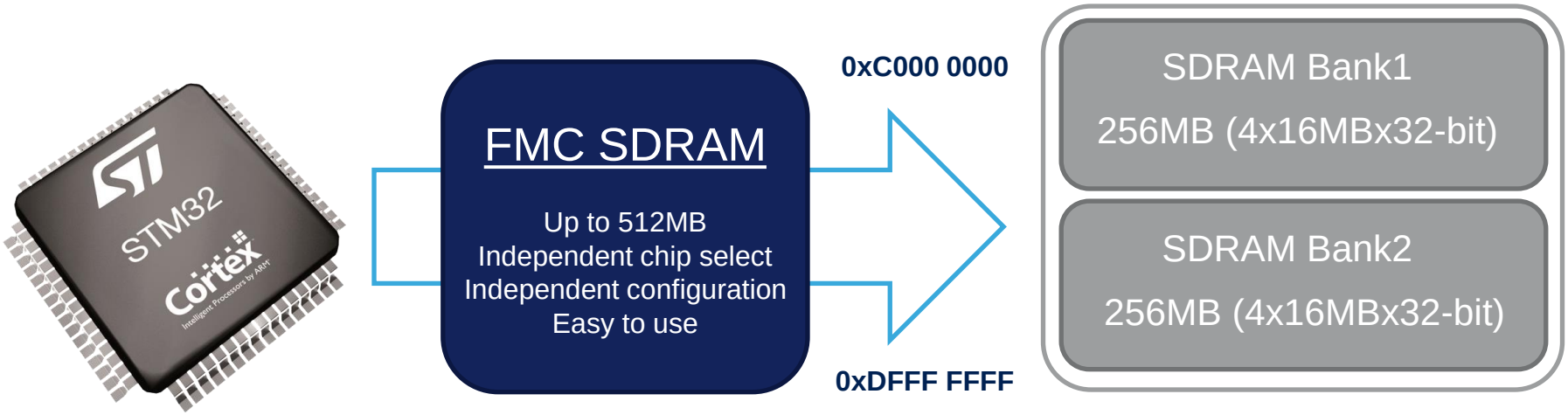
Bank is precharged (No active row)	A Row different from Row "A" is active	Row "A" is already active
• A0,A2 : 2+2+1 = 5 Cycles • B3 : 2+2+2= 6 Cycles • A1 : 2+2+2= 6 Cycles	• A0,A2 : 2+2+2+1 = 7 Cycles • B3 : 2+2+2= 6 Cycles • A1 : 2+2+2= 6 Cycles	• A0,A2 : 2+1 = 3 Cycles • B3 : 2+2+2= 6 Cycles • A1 : 2+2+2= 6 Cycles

STM32 FMC controller

life.augmented

SDRAM main features (1/4)

- Up to 512MB continues memory range split into two banks, can be seen as a single device.

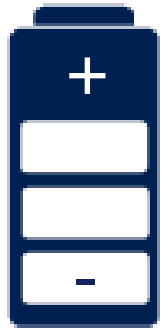


SDRAM main features (2/4)

- Fully programmable SDRAM interface
- Configurable SDRAM clock speed
  - Half AHB speed (HCLK /2),
  - One-third AHB speed (HCLK /3)
- Programmble Timing parameters for different SDRAM devices requirements
  - Row to column delay (TRCD)
  - Self refresh time
  - CAS latency of 1,2,3
- Memory data bus width : 8-bit, 16-bit and **32-bit**
- Up to 4 internal banks with configurable Row and Column sizes :
  - up to 13-bits Address Row,
  - up to 11-bits Address Column.

SDRAM main features (3/4)

- Optimized initialization sequence by software
  - The **initialization command sequence** are executed **simultaneously** for the two banks. Initialization time can be divided by 2.
- Automatic Refresh operation with programmable Refresh rate
- Energy-saving capabilities : two low power modes are supported:
  - Self-refresh Mode
  - Power-down Mode

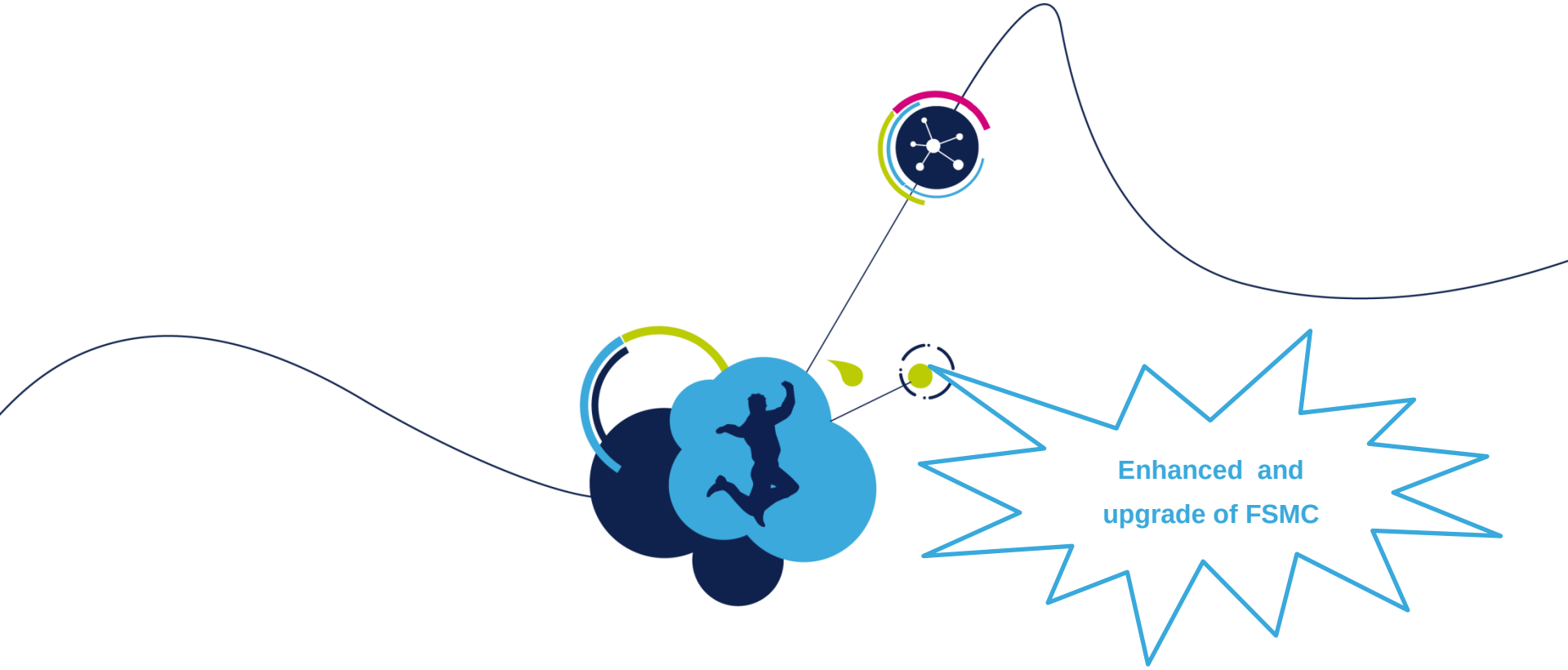


SDRAM main features (4/4)

- **Multibank ping-pong** access (FMC SDRAM controller keeps track of the active row in each bank)
- Automatic row and bank boundary management
- Optimized Read access :
  - **Cacheable Read FIFO** with depth of 6 lines x 32-bit
  - With 6x 14-bit Address Tag to identify each FIFO line content
  - Configurable Read Burst (to anticipate next read accesses during CAS latencies)
- Buffered write access
  - Write Data FIFO with depth of 16
  - With 16x Write Address FIFO to identify each FIFO line destination

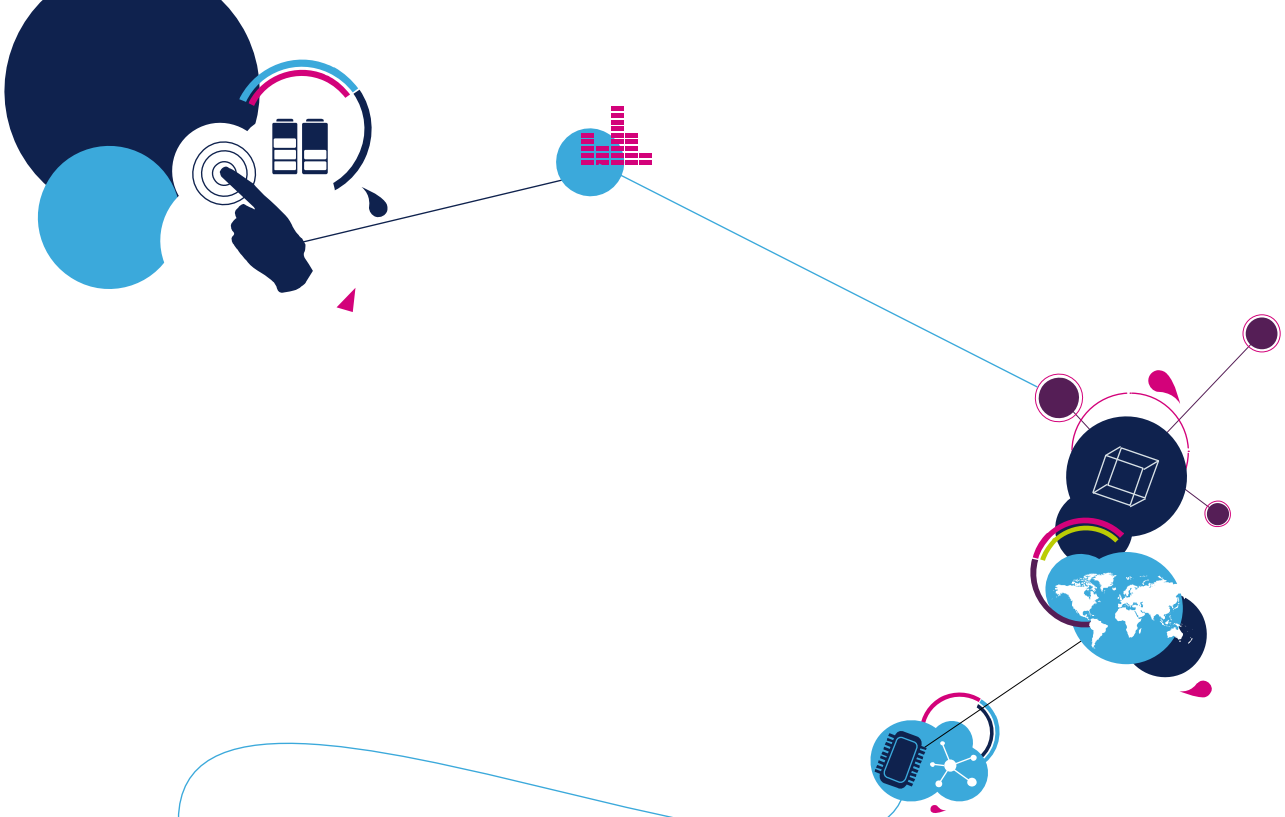
SDRAM controller benefits

- AHB Slave interface up to 180 MHz
- FMC_CLK up to 90MHz
- Grant more RAM resources for user application
- Accessible by all AHB masters
  - CPU can execute code from SDRAM
- Reduce RAM memory cost (SDRAM vs. SRAM)



Flexible Memory Controller (FMC)

life.augmented



STemWin

life.augmented

Contents

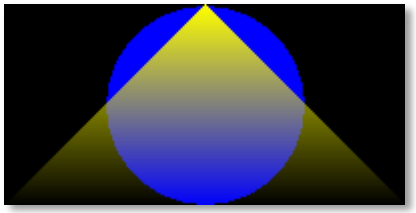
Objectives

- Provide a high level graphics library which:
  - Is easy to use
  - Is flexible to customize
  - Provides graphic objects from simple 2D elements to complex window driven objects
  - You can use the ST LTDC and Chrome-ART HW features without detailed knowledge
  - Is the industry standard in embedded field
  - Suits projects of every size → **STemWin is free of charge!**

Contents

Basic 2D library

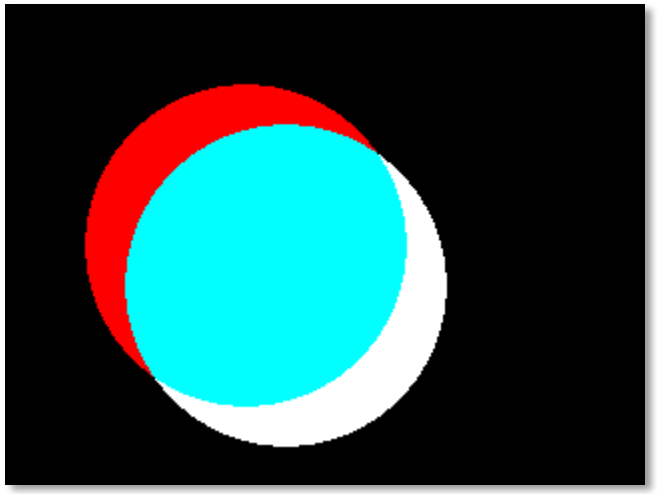
- With STemWin you can easily draw basic vector objects as
  - Lines, rectangles, arcs, polygons, graphs, bitmaps, JPEGs, PNGs and more
- Objects are drawn by a foreground color
  - `GUI_SetColor(GUI_RED);`
- STemWin supports Alpha blending – objects can overlay each other.



- Text and values can be easily written on the screen

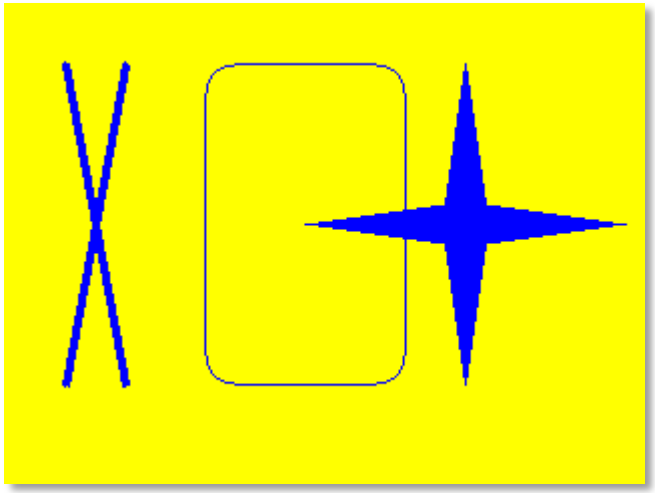
Basic 2D library - examples


```
GUI_SetColor(GUI_RED);
GUI_SetDrawMode(GUI_DRAWMODE_NORMAL);
GUI_FillCircle(120, 120, 80);
GUI_SetDrawMode(GUI_DRAWMODE_XOR);
GUI_FillCircle(140, 140, 80);
```



Basic 2D library - examples


```
GUI_SetBkColor(GUI_YELLOW);
GUI_Clear();
GUI_SetColor(GUI_BLUE);
GUI_SetPenSize(4);
GUI_DrawLine(30, 30, 60, 190);
GUI_DrawLine(30, 190, 60, 30);

GUI_DrawRoundedRect(100, 30, 200, 190, 15);

{
    const GUI_POINT aPoints[8] = {
    { 230, 30},{ 240, 100},
    { 310, 110}, { 240, 120},
    { 230, 190}, { 220, 120},
    { 150, 110}, { 220, 100},
    };
    GUI_FillPolygon(&aPoints, 8, 0, 0);
}
```



Basic 2D library - examples


```
GUI_DrawPie(100, 100, 50, 0, 60, 0);
GUI_DrawPie(100, 100, 50, 120, 180, 0);
GUI_DrawPie(100, 100, 50, 240, 300, 0);

GUI_DrawArc(200, 200, 40, 0, 0, 180);
GUI_DrawArc(200, 200, 50, 0, 30, 90);
GUI_DrawArc(200, 200, 60, 0, 30, 90);
GUI_DrawArc(200, 200, 70, 0, 30, 90);
```

Basic 2D library - Antialiasing

- Antialiasing smooths curves and diagonal lines by "blending" the background color with that of the foreground.
  - Text
    - Font converter is required for creating AA fonts.
  - Circles
  - Arcs
  - Lines
  - Polygons

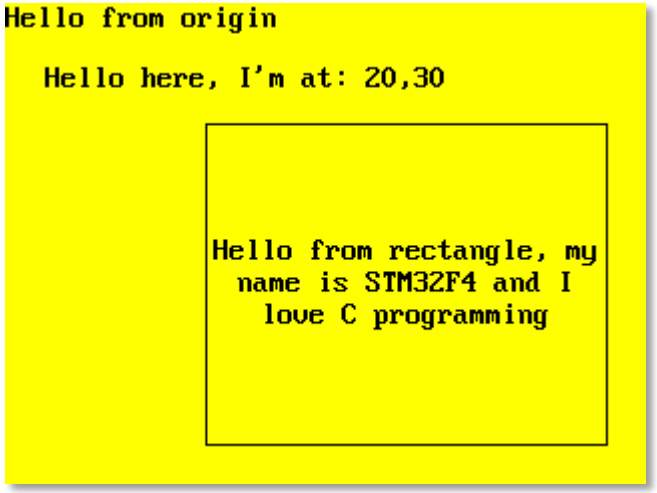
```
GUI_AA_SetFactor(4);
GUI_AA_DrawArc(160, 200, 80, 0, 0, 180);
GUI_AA_DrawArc(160, 200, 90, 0, 30, 90);
GUI_AA_DrawArc(160, 200, 100, 0, 30, 90);
GUI_AA_DrawArc(160, 200, 110, 0, 30, 90);
```

The higher the number of shades used between background and foreground colors, the better the antialiasing result (and the longer the computation time).

Basic 2D library – text

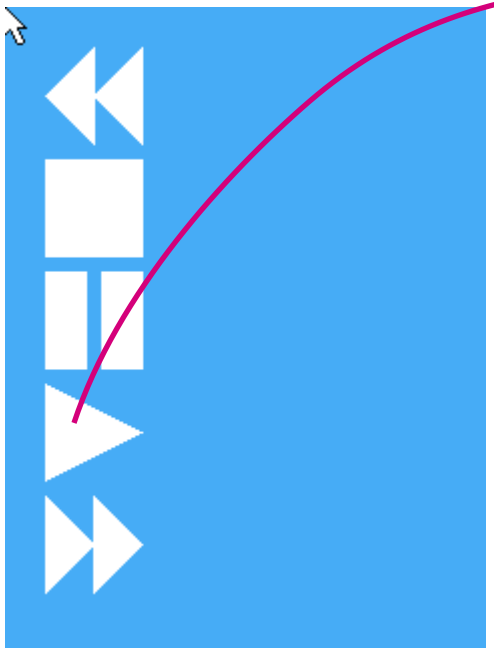
- STemWin enables you to add text of any font into your GUI
- Several API functions are available to ease of use for text
  - Display text at specific position
  - Manage text inside a rectangle

```
GUI_SetFont(&GUI_Font8x16);
GUI_DispString("Hello from origin");
GUI_DispStringAt("Hello here, I'm at:
20,30", 20,30);
{
  GUI_RECT pRect = {100, 60, 300, 220};
  GUI_DrawRect(100, 60, 300, 220);
  GUI_DispStringInRectWrap("Hello from
  rectangle, my name is STM32F4 and I love
  C programming", &pRect, GUI_TA_VCENTER |
  GUI_TA_HCENTER, GUI_WRAPMODE_WORD);
}
```



Example **PLAYER** application

- Let's use the gained knowledge to make a simple application GUI
- With basic 2D object we can start to draw vector icons



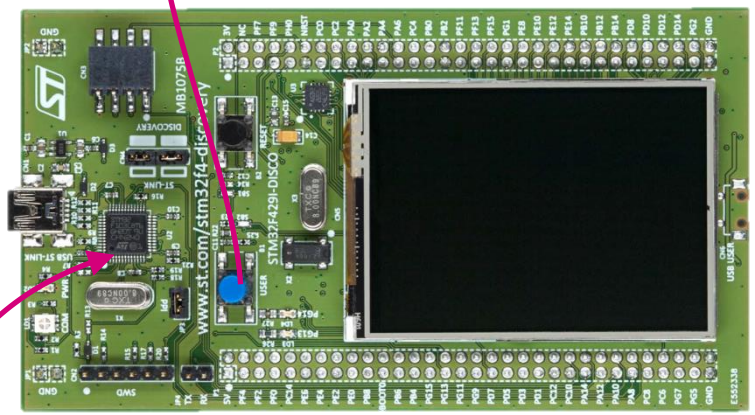
```
void DrawPlay(int x, int y, int size)
{
    GUI_POINT pPoint[3];
    pPoint[0].x = 0;
    pPoint[0].y = 0;
    pPoint[1].x = size;
    pPoint[1].y = size / 2;
    pPoint[2].x = 0;
    pPoint[2].y = size;

    GUI_FillPolygon(pPoint, 3, x, y);
}
```

- Other icons can be easily drawn very similar to **PLAY** icon

Example **PLAYER** application

- You can see the application in the Discovery kit. Pushing the user button you can navigate to next steps of this example application.

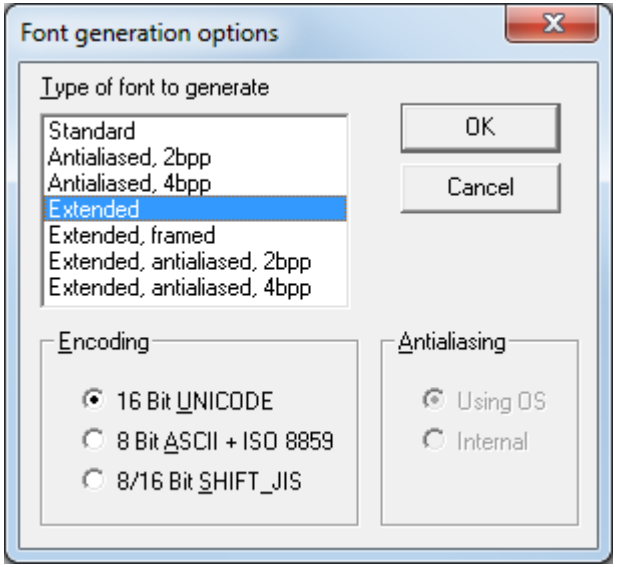
- Move forward by pushing the USER button

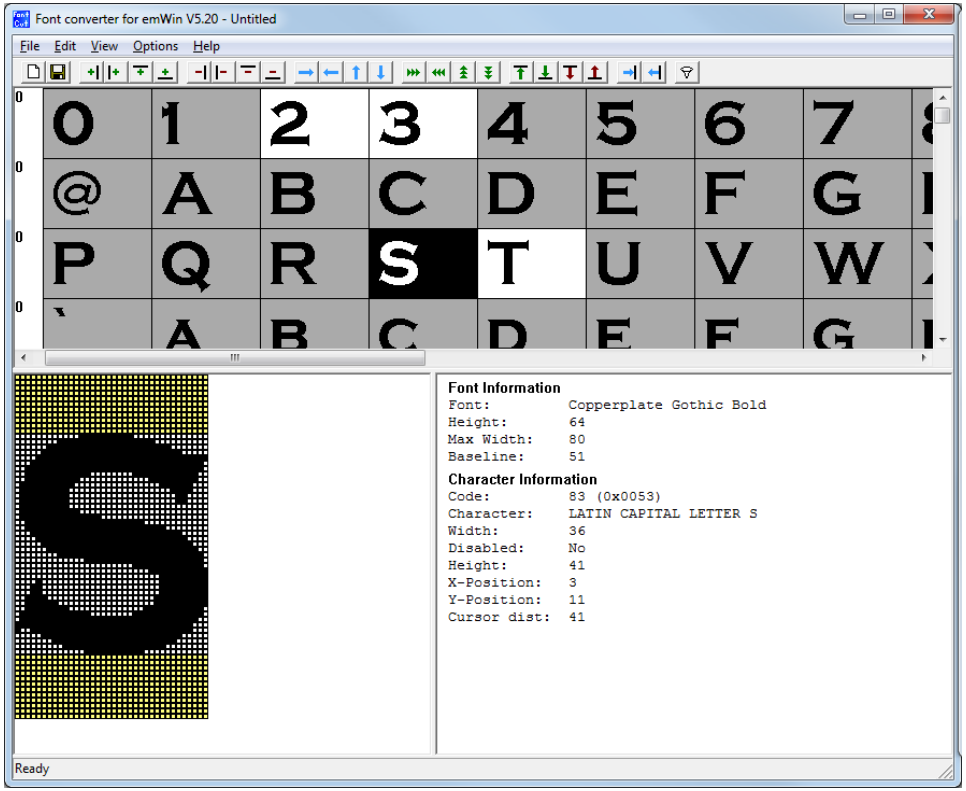


- Embedded STLINK debugger

Basic 2D library – Font convertor SW

- You can manage your fonts in-application
  - Create fonts from **any Windows font**
  - Mange the number of characters so that you **save only those you need** → safe ROM
  - Export as .c files





life.augmented

Basic 2D library – using generated font

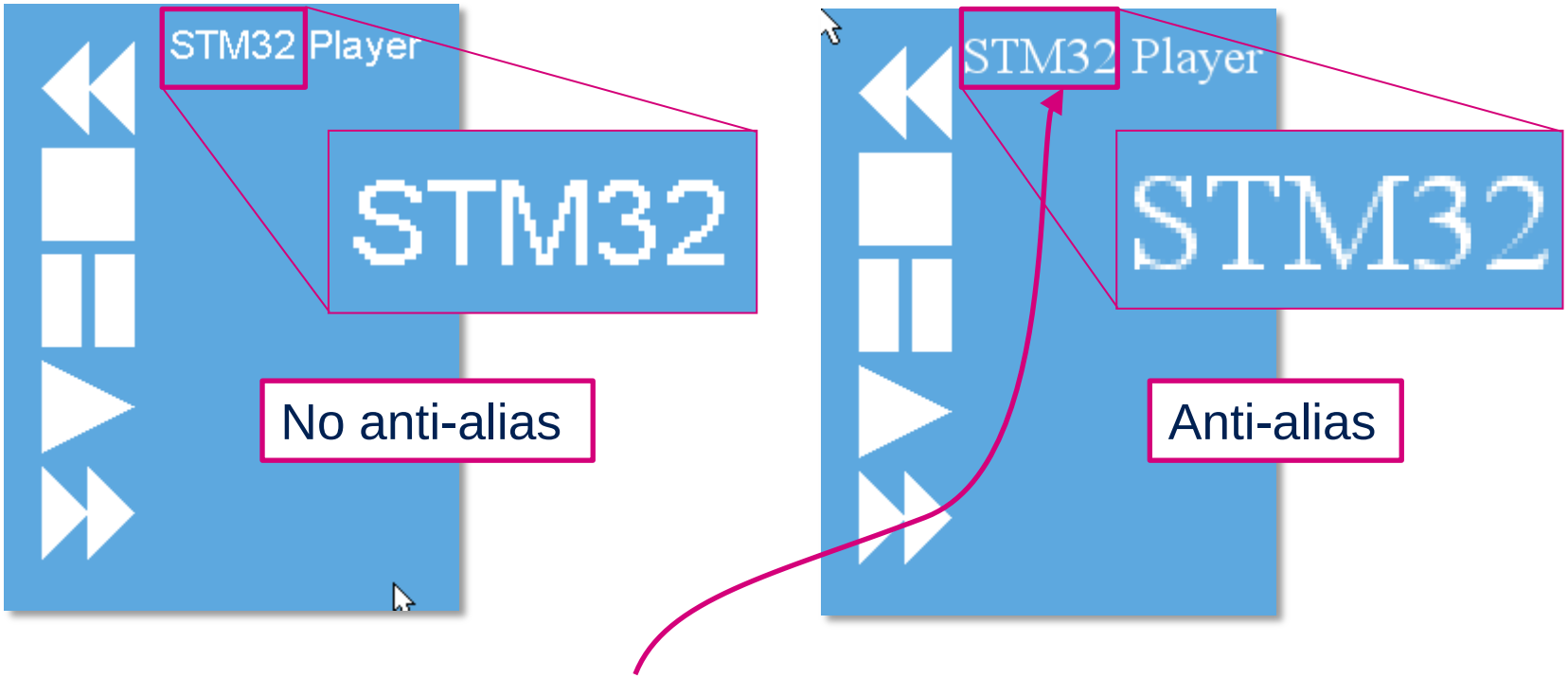
- Using the font we have generated is very easy
  - Include the generated .c file into the project
  - Include the external font declaration to all modules which will use it
  - Use **GUI_SetFont()** function to point STemWin to this font

```
extern GUI_CONST_STORAGE GUI_FONT GUI_FontCopperplateGothicBold64;
extern GUI_CONST_STORAGE GUI_FONT GUI_FontCopperplateGothicBold64_aa;
..
GUI_RECT pRect1 = {0, 60, 319, 119};
GUI_RECT pRect2 = {0, 120, 319, 180};
GUI_SetFont(&GUI_FontCopperplateGothicBold64);
GUI_DispStringInRect("STM32", &pRect1, GUI_TA_VCENTER | GUI_TA_HCENTER);
GUI_SetFont(&GUI_FontCopperplateGothicBold64_aa);
GUI_DispStringInRect("STM32", &pRect2, GUI_TA_VCENTER | GUI_TA_HCENTER);
```

Drawing anti-aliased text takes much more time! (and memory as well)

Example **PLAYER** application

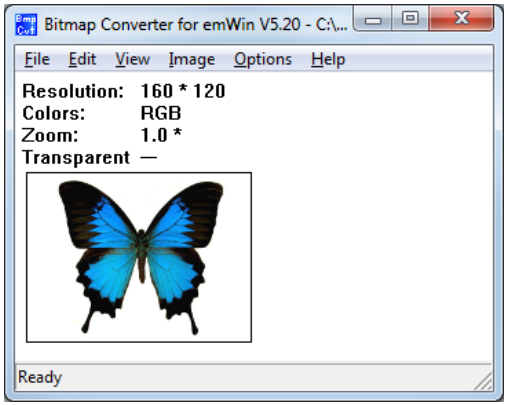
- Now we can add some text to our application




```
extern GUI_CONST_STORAGE GUI_FONT GUI_FontTimesNewRoman31;
GUI_SetFont(&GUI_FontTimesNewRoman31);
GUI_DispStringInRect("STM32 Player", &rect, GUI_TA_CENTER);
```

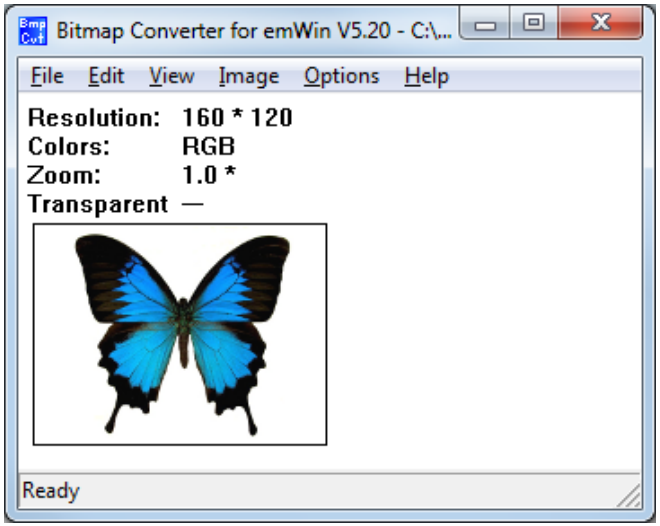
Basic 2D library bitmaps

- Static bitmaps or streamed bitmaps are supported
  - **Static bitmap** is completely available during drawing (e.g. stored in Flash memory)
  - **Streamed bitmaps** are available only by parts received by the MCU (e.g. reception from data card).
- Supported formats
  - Internal bitmap format
  - BMP
  - JPEG
  - PNG
  - GIF

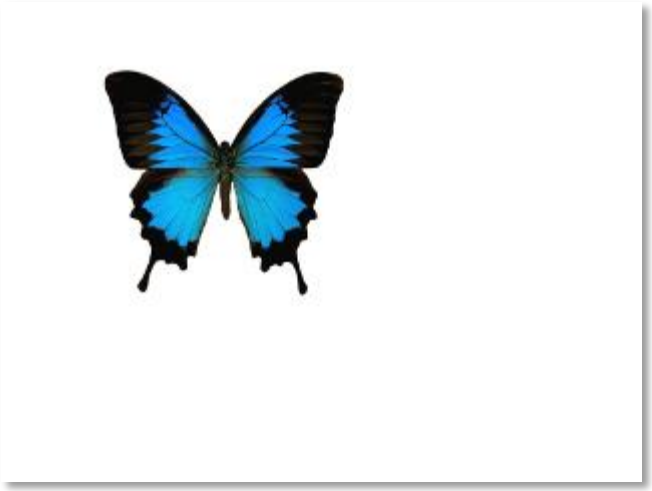


- **BmpCvt.exe** can be used for bitmap conversion and storage to .c files
- **Bin2c.exe** can be used to store any binary in .c form, e.g. complete bitmap or jpeg files

Basic 2D library – bitmaps example




```
extern GUI_CONST_STORAGE GUI_BITMAP bmbutterfly;
GUI_DrawBitmap(&bmbutterfly, 30, 30);
```

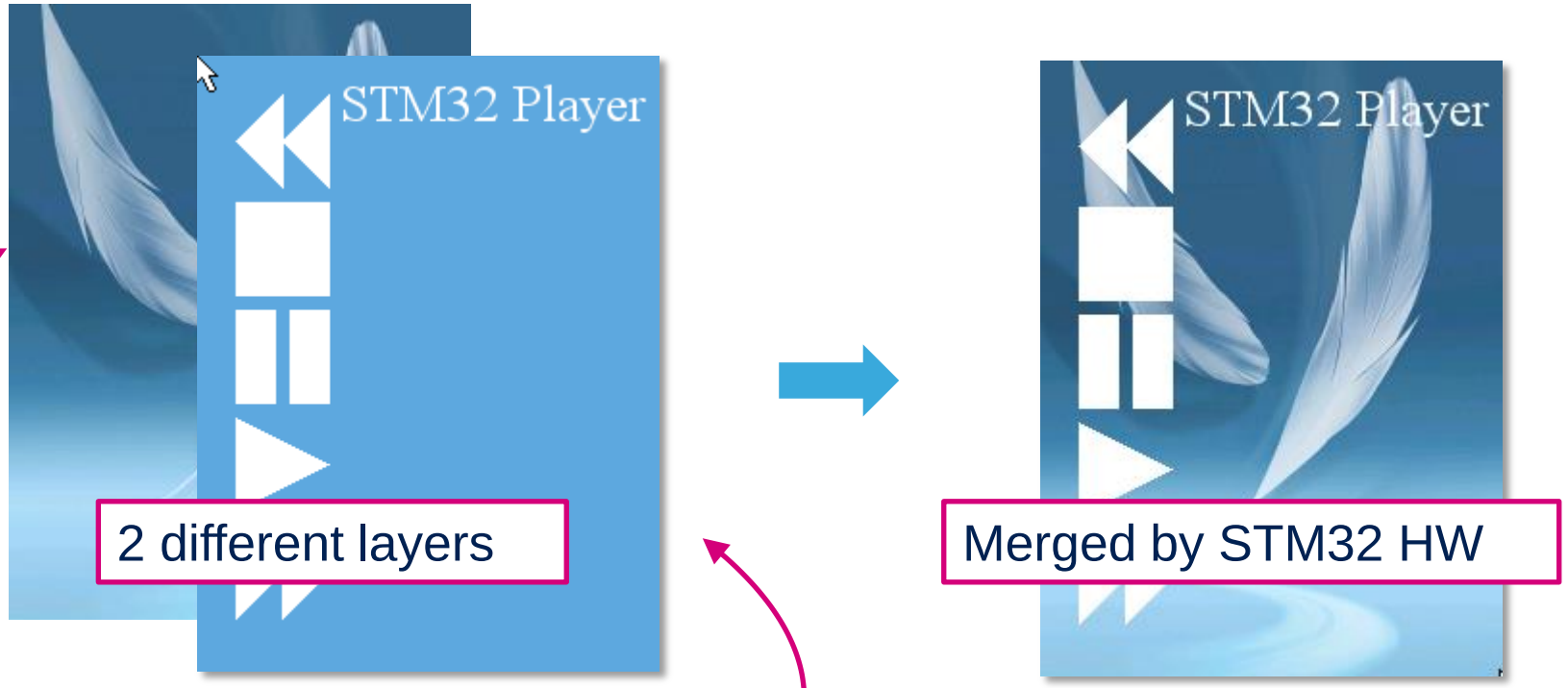


Storing a bitmap in the same format as the LCD increases drawing performance → no pixel conversion needed, but may need more storage space.

life.augmented

Example **PLAYER** application

- What about some nice background behind our player?




```
extern GUI_CONST_STORAGE GUI_BITMAP bmbluegirl565;
GUI_SelectLayer(0); // select the background layer
GUI_DrawBitmap(&bmbackground, 0, 0);
GUI_SelectLayer(1);
// continue with drawing foreground
```

life.augmented

Human interface

Communication between the application and the user is mostly done by keyboard and/or **P**ointer **I**nput **D**evices or touch devices. The following functions are available:

- **GUI_StoreKeyMsg()**
  - If a keyboard event occurs (pressing or releasing a key) it should be passed to this routine.
- **GUI_PID_StoreState()**
  - If a PID event occurs (pressed, released or changing the position) it should be passed to this routine.
- **GUI_TOUCH_StoreState()**
  - In case of human touch on touchscreen, this function should be called

The WM then automatically polls the keyboard and the PID buffer. Keyboard input will be passed to the currently focused window and PID input to the respective window

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 hierarchy 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**

WM – callback mechanism

The callback mechanism requires a callback routine for each window. These routines have to support the following:

- **Painting the window**
  - Each window has to draw itself. This should be done when receiving a **WM_PAINT** message.
- **Default message handling**
  - Plain windows need to call the function `WM_DefaultProc()` to avoid undefined behavior of the window.

Further the WM needs to 'stay alive'. This can be done within a simple loop after creating the windows. It has nothing to do but calling `GUI_Delay()` or `GUI_Exec()` which does the following:

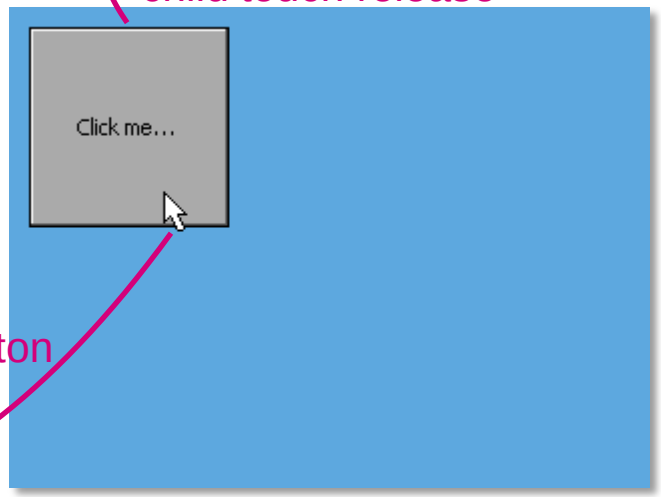
- **PID management**
- **Key input management**
- **Timer management**

```
hButton = BUTTON_Create( 10, 10, 100, 100, GUI_ID_BUTTON0, WM_CF_SHOW);
BUTTON_SetText(hButton, "Click me...");
WM_SetCallback(WM_HBKWIN, myCbBackgroundWin);
WM_SetCallback(hButton, myCbButton);
... ... ...
static void myCbBackgroundWin(WM_MESSAGE *pMsg) {
  int NCode, Id;
  switch (pMsg->MsgId) {
  case WM_NOTIFY_PARENT:
    Id    = WM_GetId(pMsg->hWinSrc);      /* Id of widget */
    NCode = pMsg->Data.v;                 /* Notification code */
    if ((Id == GUI_ID_BUTTON0) && (NCode == WM_NOTIFICATION_RELEASED))
      buttonClicked = 0;
   break;
   case WM_PAINT:
      GUI_SetBkColor(STBLUE);
      GUI_Clear();
   break;
  }
}
... ... ...
static void myCbButton(WM_MESSAGE *pMsg) {
   switch (pMsg->MsgId) {
   case WM_TOUCH:
   if (((GUI_PID_STATE*)(pMsg->Data.p))->Pressed == 1)
      buttonClicked = 1;
      break;
   case WM_SIZE:
   // add some code
   break;
   default:
      BUTTON_Callback(pMsg);
   }
}
```

Initialization – called only once!

Redraw part

On release WM calls parent callback function with message informing about child touch release



On click WM calls Button callback function

Default callback must be called to achieve proper functionality

life.augmented

WM – Widget library

Widget = **Wi**ndow + Ga**dget**

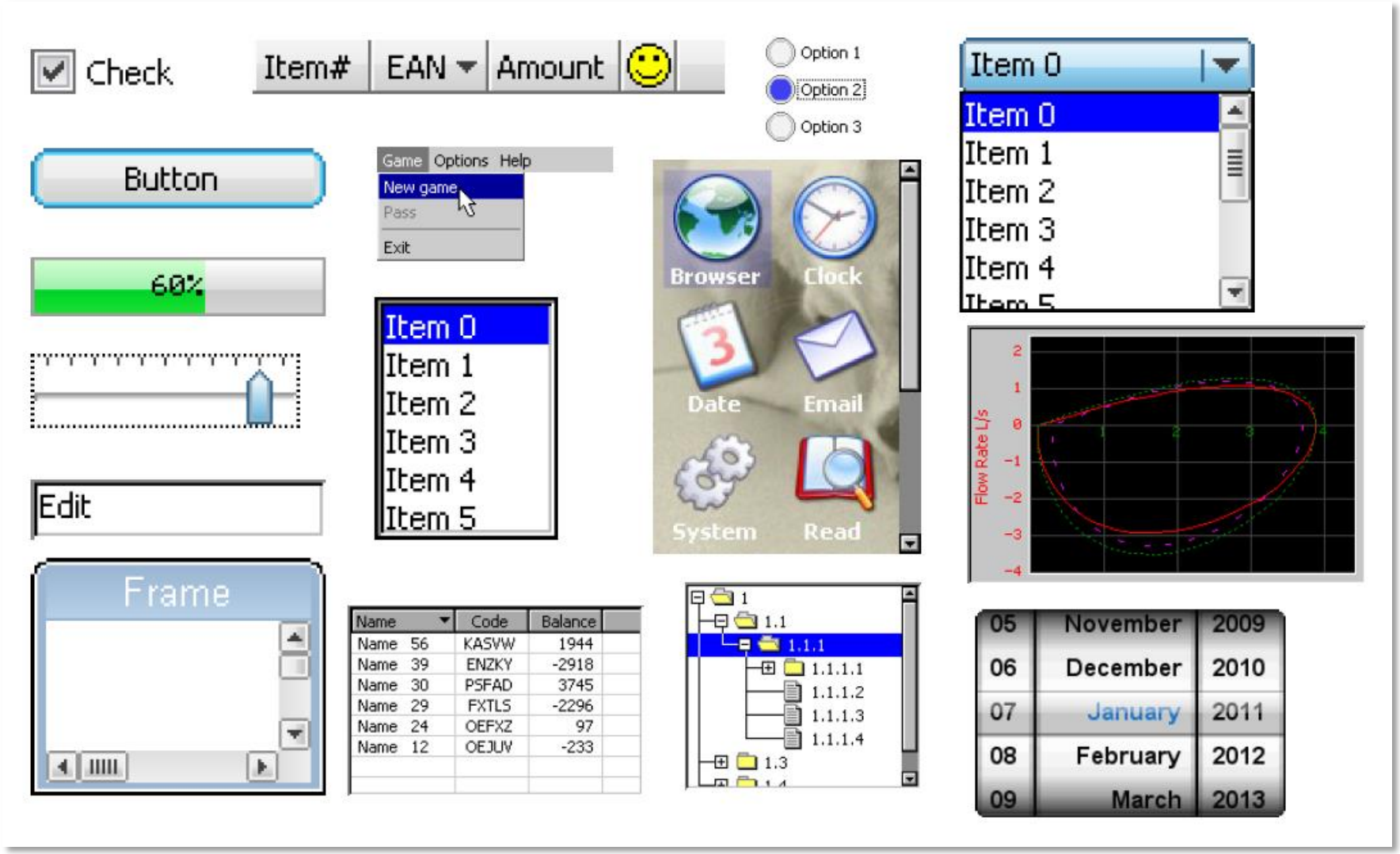
Currently the following widgets are supported:

- 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**
    A widget only needs to be created indirectly if it is to be included in a dialog box.
    <WIDGET>_CreateIndirect() // Creates a widget to be used in dialog boxes.

WM – Widget library



life.augmented

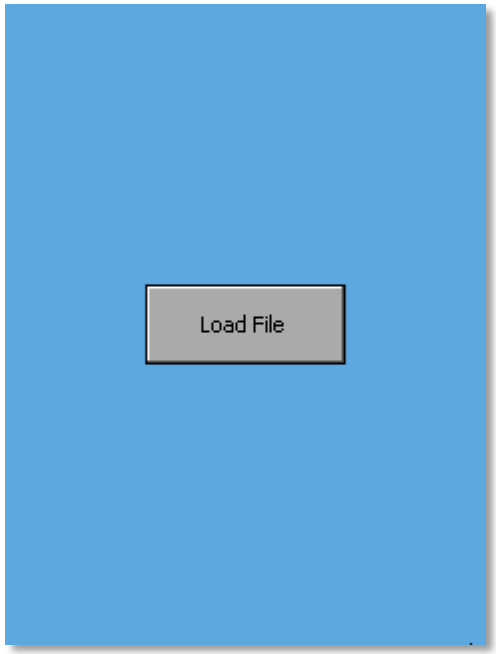
Example **PLAYER** application

- Let's rework our PLAYER example using WM.
- Start with single button for prompt the user to select song

```
#include "BUTTON.h"
static BUTTON_Handle hButton;

hButton = BUTTON_Create( 320/2 - BUTTON_WIDTH / 2, 240/2 -
BUTTON_HEIGHT / 2, BUTTON_WIDTH, BUTTON_HEIGHT, GUI_ID_OK,
WM_CF_SHOW);
BUTTON_SetText(hButton, "Load File");

GUI_Exec();
```

Always include the widget header

Only after call of this function the windows get paint



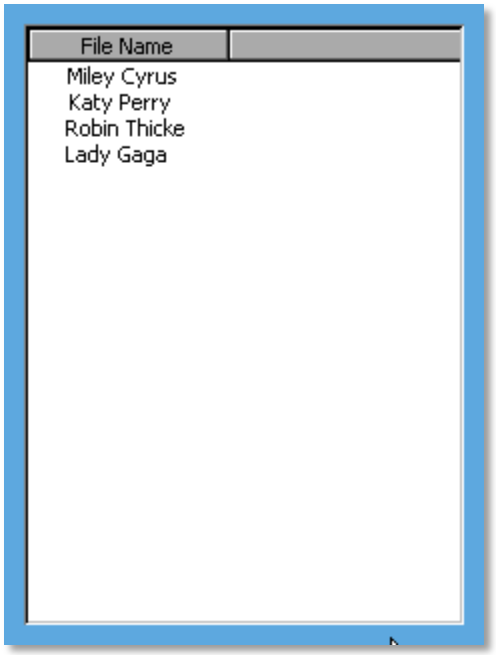
life.augmented

Example **PLAYER** application

- After click on button show a LISTVIEW widget with selection of songs

```
hListView = LISTVIEW_Create(320/2 - LISTVIEW_WIDTH / 2, 240/2 -
LISTVIEW_HEIGHT / 2, LISTVIEW_WIDTH, LISTVIEW_HEIGHT, WM_HBKWIN,
GUI_ID_LISTVIEW0, 0, 0);

LISTVIEW_AddColumn(hListView, 100, "File Name", GUI_TA_CENTER);
for (i = 0; i < GUI_COUNTOF(myFileTable); i++) {
   LISTVIEW_AddRow(hListView, myFileTable[i]);
}
```

In this loop just copy text array into the widget



Example **PLAYER** application

- Now we can use ICONVIEW widget to change vector icons by nice semi-transparent bitmap icons

- Simply use icons in .png, convert them to .c by bitmap converter, add to project and use as ICONVIEW widget source:

```
hIconView = ICONVIEW_CreateEx(ICONS_HOR_POS, ICONS_VER_POS, LCD_GetXSize() -
2*ICONS_HOR_POS, bmStyleNexticon1.YSize + GUI_Font8_1.YSize + 10, WM_HBKWIN,
WM_CF_SHOW, ICONVIEW_CF_AUTOSCROLLBAR_V, GUI_ID_ICONVIEW0,
bmStyleNexticon1.XSize, bmStyleNexticon1.YSize + GUI_Font8_1.YSize);
...
for (i = 0; i < iconNumber; i++) {
    ICONVIEW_AddBitmapItem(hIconView, myBitmapItem[i].pBitmap,
myBitmapItem[i].pText);
}
```

In this loop just copy pointers to icon bitmaps and texts into the widget

Text may appear also as widget TEXT

```
hTextSong = TEXT_CreateEx (0, 200, 320 - 1, 30,
WM_HBKWIN, WM_CF_SHOW, 0, GUI_ID_TEXT0,
pSongString);
```

life.augmented

Example **PLAYER** application

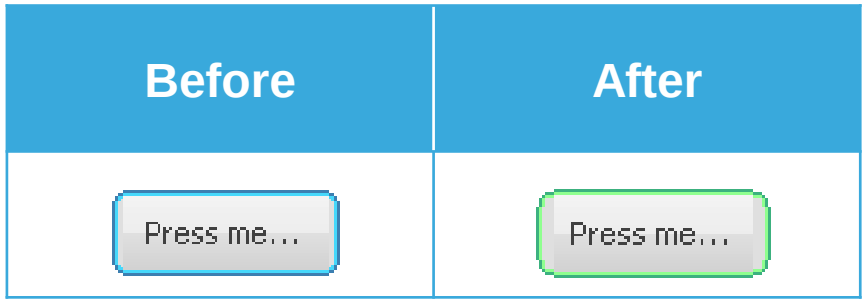
- Finally put the bitmap background into background layer
- To do so, we only need to switch to LCD background layer
  - In reality this only changes the frame buffer start address for drawing operations→ no performance lost
- Then draw bitmap as done before

```
GUI_SelectLayer(0); // select background layer
GUI_DrawBitmap(&bmbackground, 0, 0);
GUI_SelectLayer(1);
```

Select back the foreground layer for all subsequent operations

WM – Skinning

- **Skinning** is a method of changing the appearance of one or multiple widgets.
- A skin is just a **simple callback function** which is available for drawing all details of a widget.
- These widgets can be skinned:
  - BUTTON
  - CHECKBOX
  - DROPDOWN
  - FRAMEWIN
  - HEADER
  - PROGBAR
  - RADIO
  - SCROLLBAR
  - SLIDER



Interfacing to HW, configuration

- STemWin is high level GUI package, but it needs a low level driver to access MCU resources

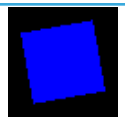
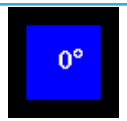
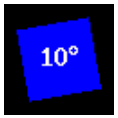
- In the package, there are 2 drivers:
  - STM32F4 **LTDC** customized driver which benefits from **Chrome-ART** acceleration for copying/fill routines
  - **FlexColor** driver used for interfacing external LCD drivers through FSMC interface. E.g. popular **Ilitek ILI9320, ILI9325 or Himax HX8347** devices.

- Configuration through LCD_Conf.c, for example:

```
#define GUI_NUM_LAYERS 2   // defines how many layers are used
#define XSIZE_PHYS 480     // set-up the physical dimensions
#define YSIZE_PHYS 272
// and the color mode for layer 0
#define COLOR_MODE_0 CM_ARGB8888
// physical address of the frame buffer
#define LCD_FRAME_BUFFER ((U32)0xC0000000)
```

Other STemWin features

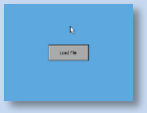
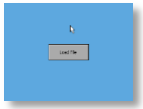
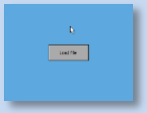
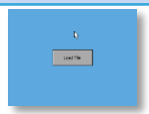
- Memory devices
  - Memory Devices can be used in a variety of situations, mainly to **prevent the display from flickering** when using drawing operations for overlapping items.
  - This requires additional RAM memory

API function	Without Memory Device	With Memory Device
Step 0: Initial state		
Step 1: GUI_Clear()		
Step 2: GUI_DrawPolygon()		
Step 3: GUI_DispString()		
Step 4: GUI_MEMDEV_CopyToLCD()		

Other STemWin features

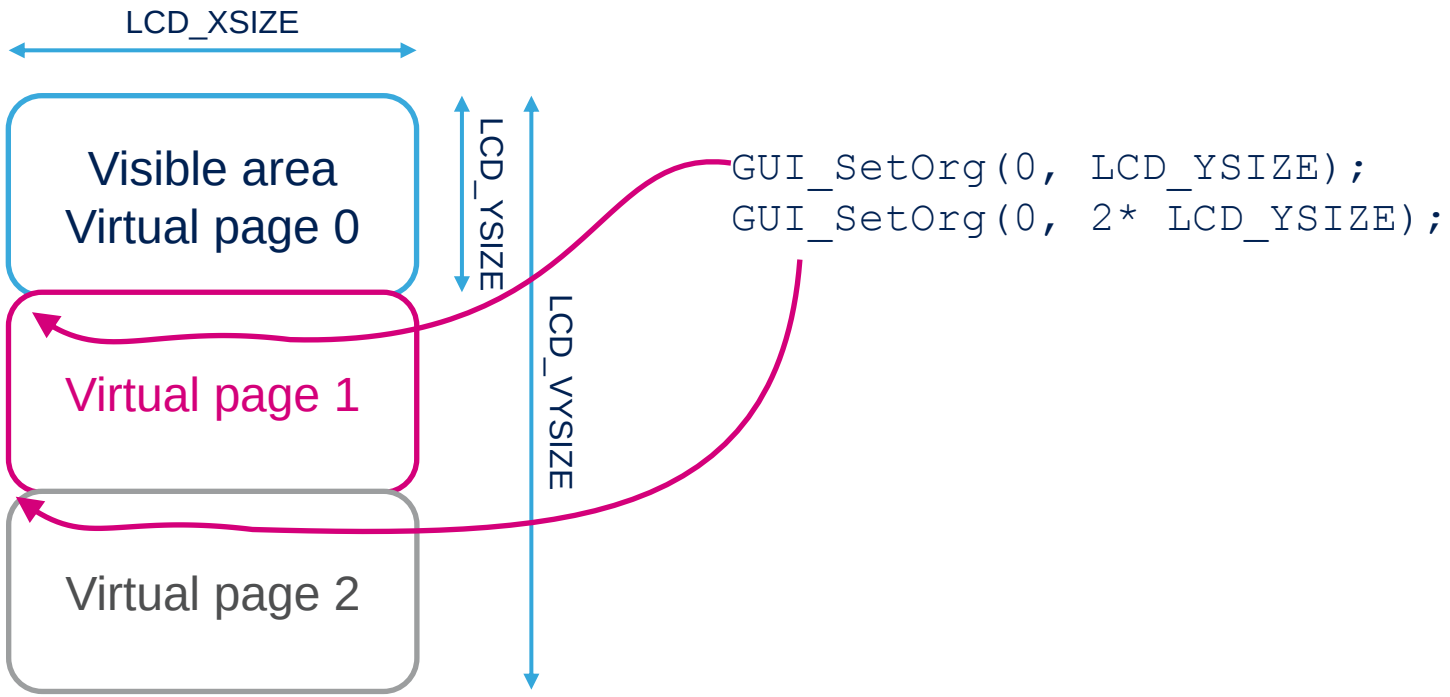
- Multiple buffering
  - With multiple buffers enabled there is a **front buffer** which is used by the display controller to generate the picture on the screen and **one or more back buffers which are used for the drawing operations.**

- In general it is a method which is able to avoid several unwanted effects:
  - The visible process of drawing a screen item by item
  - Flickering effects caused by overlapping drawing operations
  - Tearing effects caused by writing operations outside the vertical blanking period

API function	1st buffer	2nd buffer
1. Copy **active** buffer to inactive		
1. Drawing operations in inactive buffer		
3. Set 2nd buffer as **active**		

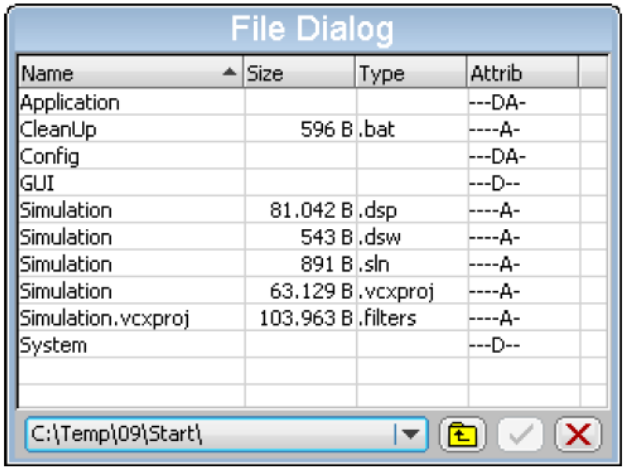
Other STemWin features

- Virtual screens
- Display area greater than the physical size of the display. It requires additional video memory and allows instantaneous switching between different screens.
- Similar to Multiple buffering, but **no copy operation** is performed.



Other STemWin features

- And many others:
  - Motion JPEG video
  - Support for **real time operating systems**
  - Dialogs
  - GUIBuilder SW
  - Sprites
  - **Arabic, Thai, Chinese, Japanese** fonts
  - VNC Server



IAR Tool Installation

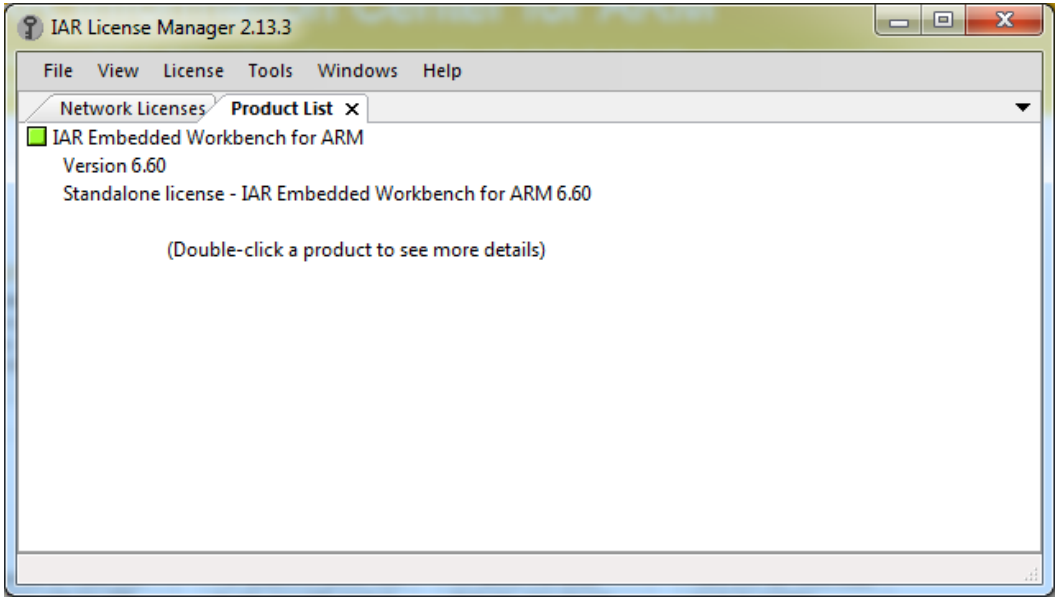
License Steps

life.augmented

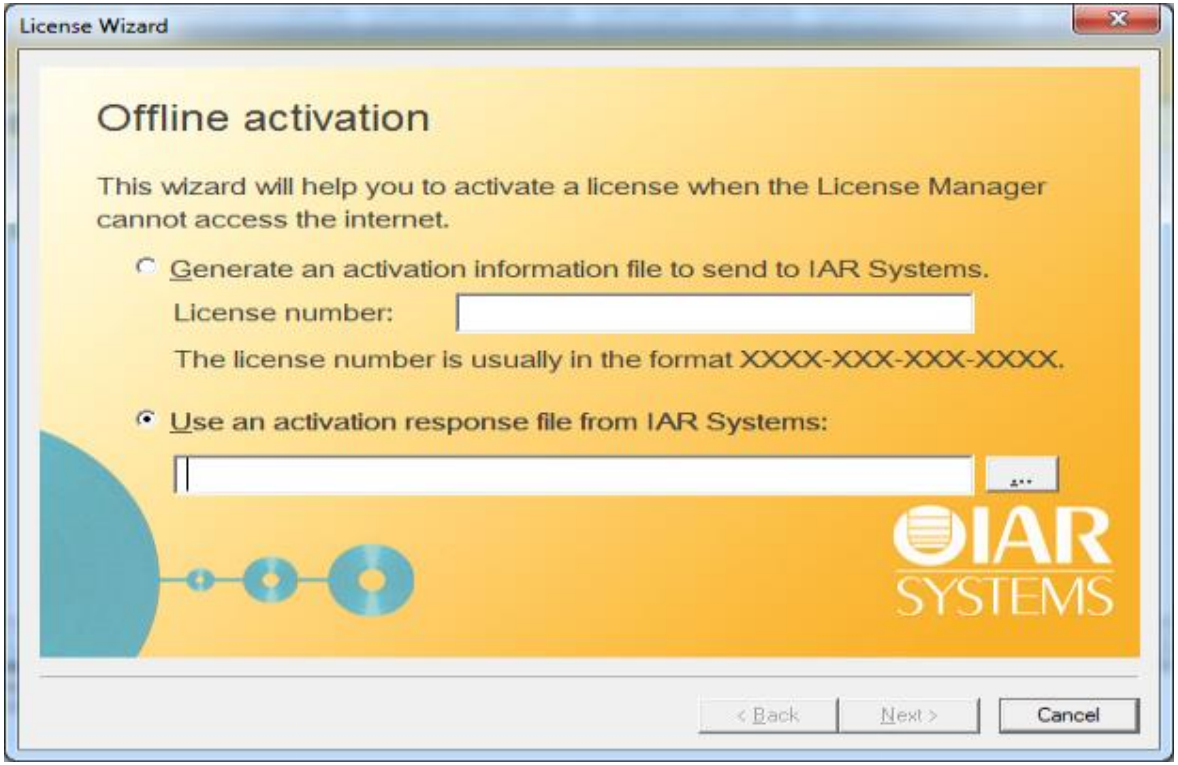
Step #5 - EWARM Installation

- Launch IAR Embedded Workbench from Start Menu
  - Start Menu > All Programs > IAR Systems > IAR Embedded Workbench for ARM 6.60 > IAR Embedded Workbench
  - Make sure this is the version you just installed if you have other older versions
  - If you see an activation window open at this stage close it

- Open the License Manager from the information Center main window by selecting:
  - Help/License Manager.



Step #6 - EWARM Installation



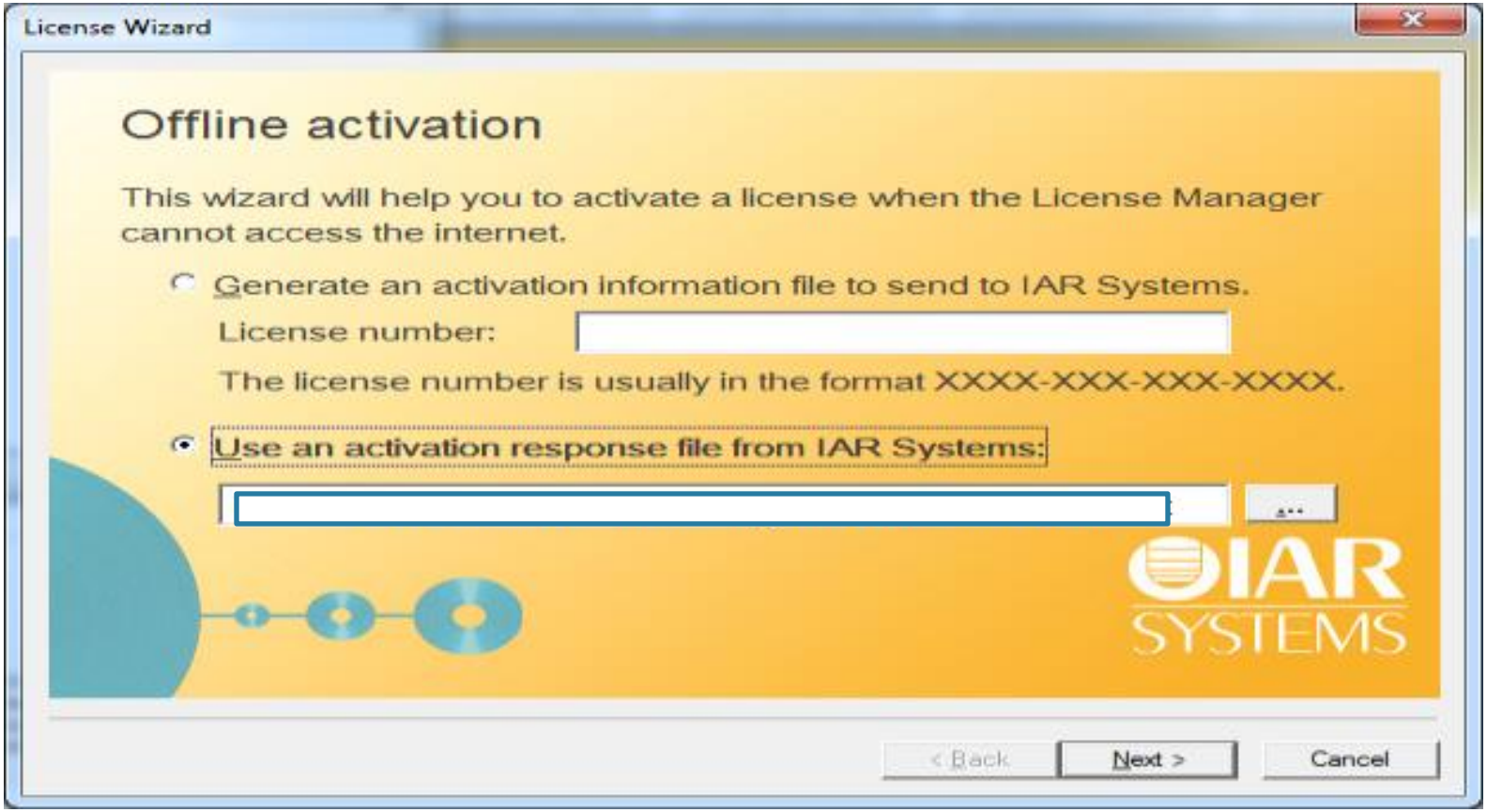
- In the license manager window open the Offline activation window by selecting:

1. License/Offline activation.

In the offline activation window select:

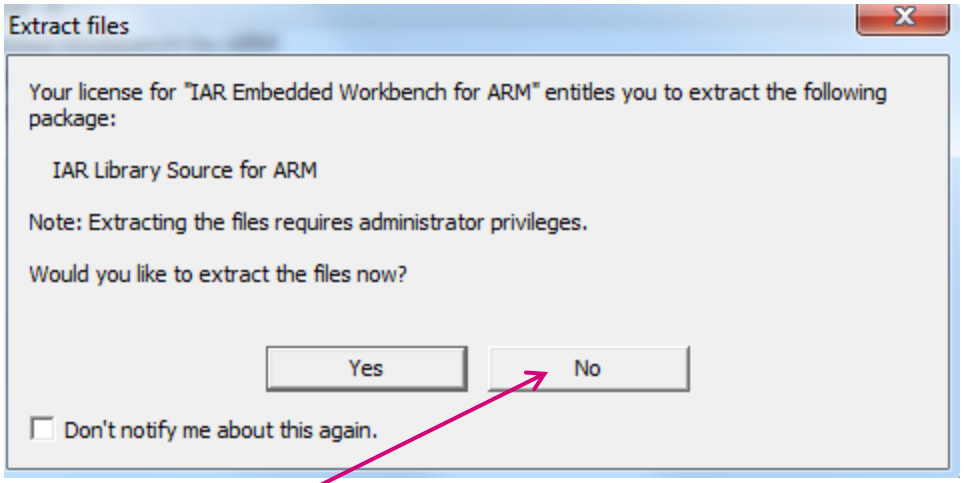
- 2. Use an activation response file from IAR systems.
- 3. Select the … bar to select your offline activation window.

Step #7 - EWARM Installation



- Select the C:\STM32Seminar\IAR_EWARM\ActivationResponse.txt and select “open”.

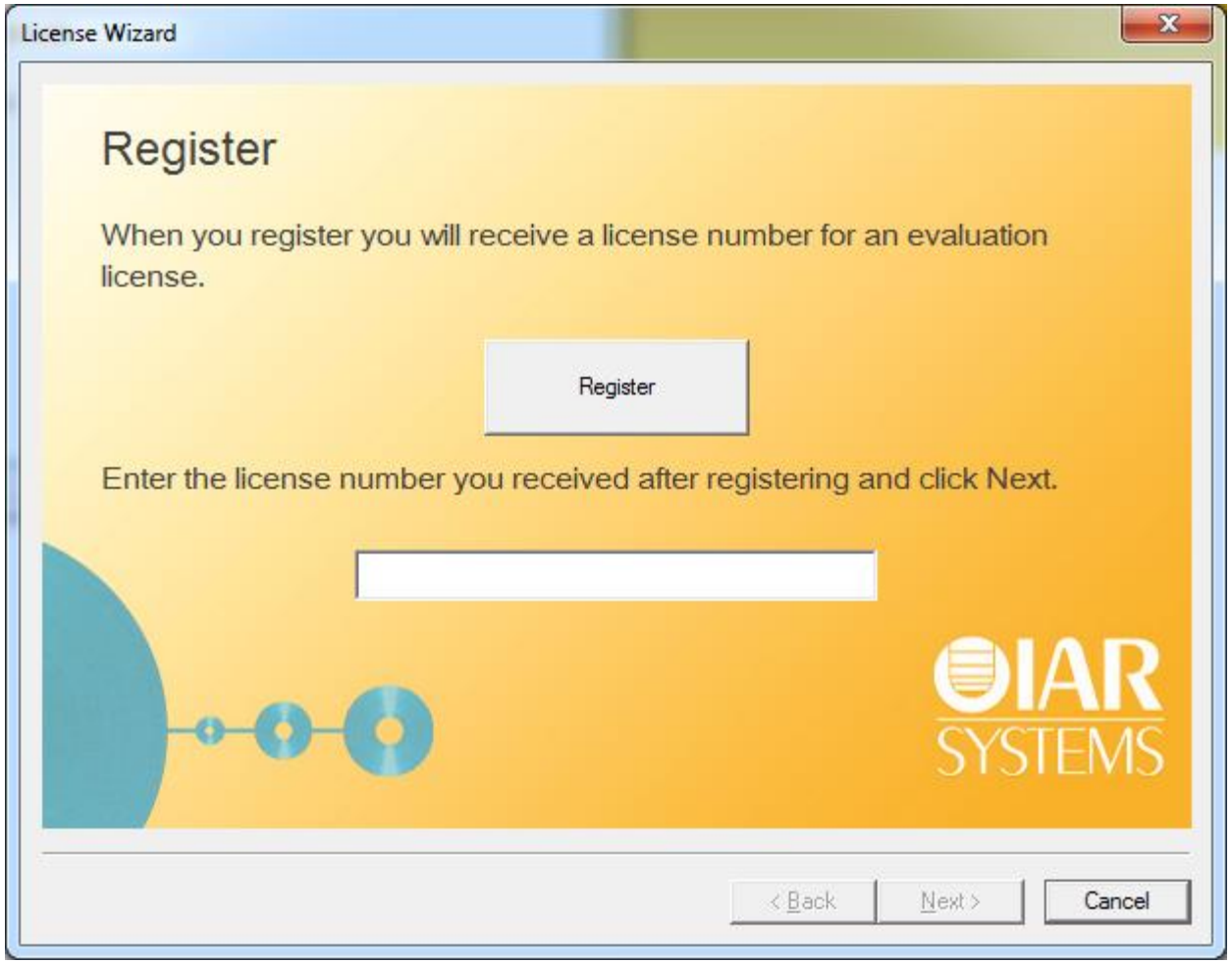
Step #9 - EWARM Installation



Single-click on

No **(you can install this later if you wish).**

To Extend the License Online after the Seminar



- Open the License Manager from the information Center main window by selecting: Help/License Manager.
- Open the “Get Evaluation License” by selecting:
- License/Get Evaluation license from the license manager.
- Select the Register and you will be prompted to IAR registration website

ST-LINK/V2
Windows® Driver Installation

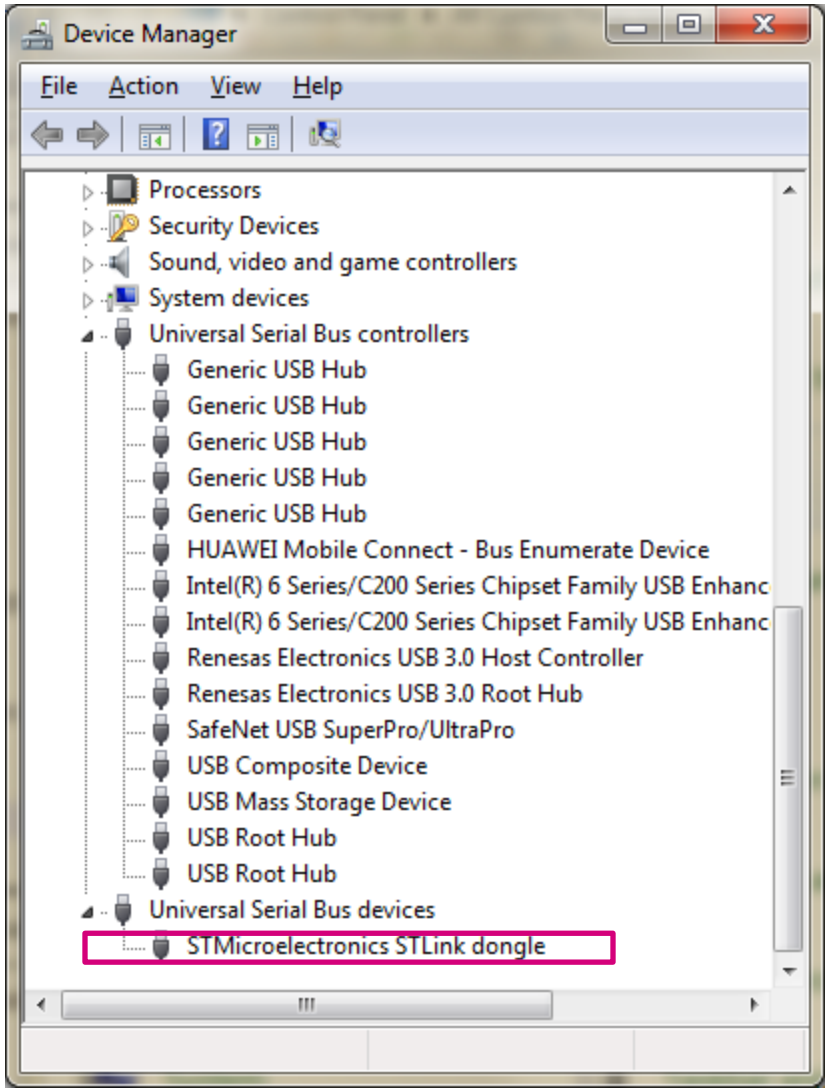
life.augmented

Install ST-Link Driver

- The STM32F429 Discovery board includes an ST-Link/V2 embedded programming and debug tool
- The driver for the ST-Link can be found at
  - C:/STM32Seminar/IAR_EWARM\STLink_SignedDriver\stlink_winusb_install.bat
- The Discovery board should be unplugged
- Right click on the file: stlink_winusb_install.bat and RUN AS ADMINISTRATOR and click through until the driver installation is complete.
  - The above steps are common to Windows 7 and Windows 8.

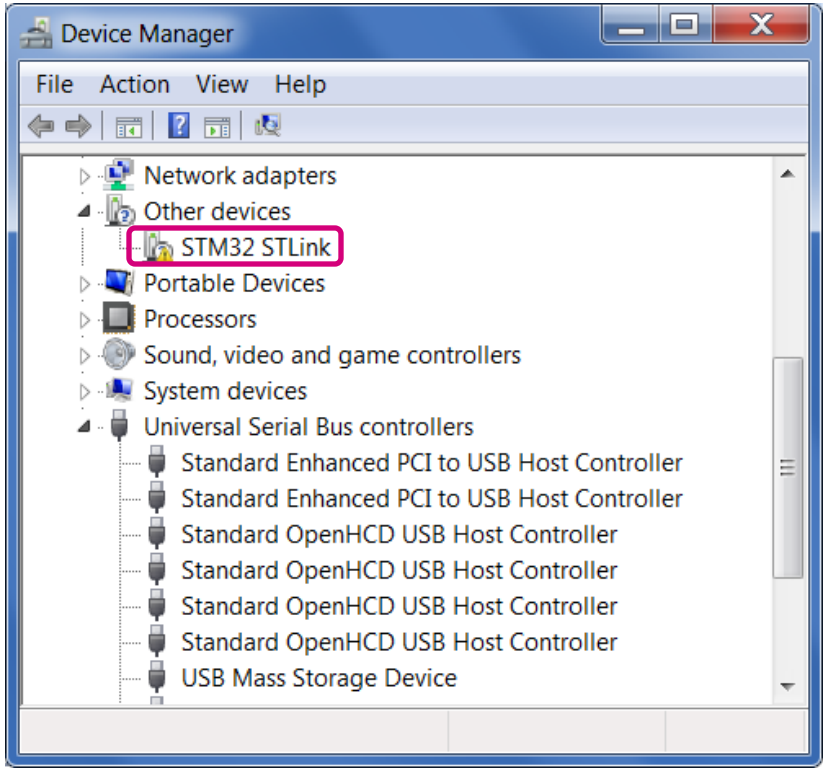
Connect the Discovery Kit, Enable ST-Link

- Using the USB cable, connect the mini-B male connector in the STM32F429I-Discovery USB port at the top of the board, and connect the USB A male connector into your laptop
- Wait for Windows to recognize the ST-Link device and follow any steps required to install the driver
- Upon successful driver recognition, the ST-Link device should be fully enumerated in the Windows Device Manager as shown:

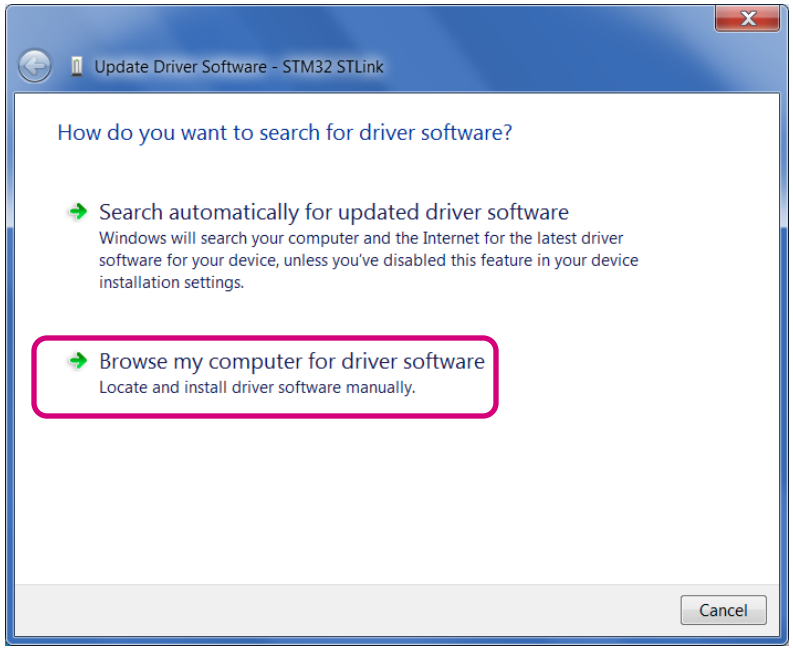


Step #1 ST-Link Driver Trouble Shooting, (Windows® 7)

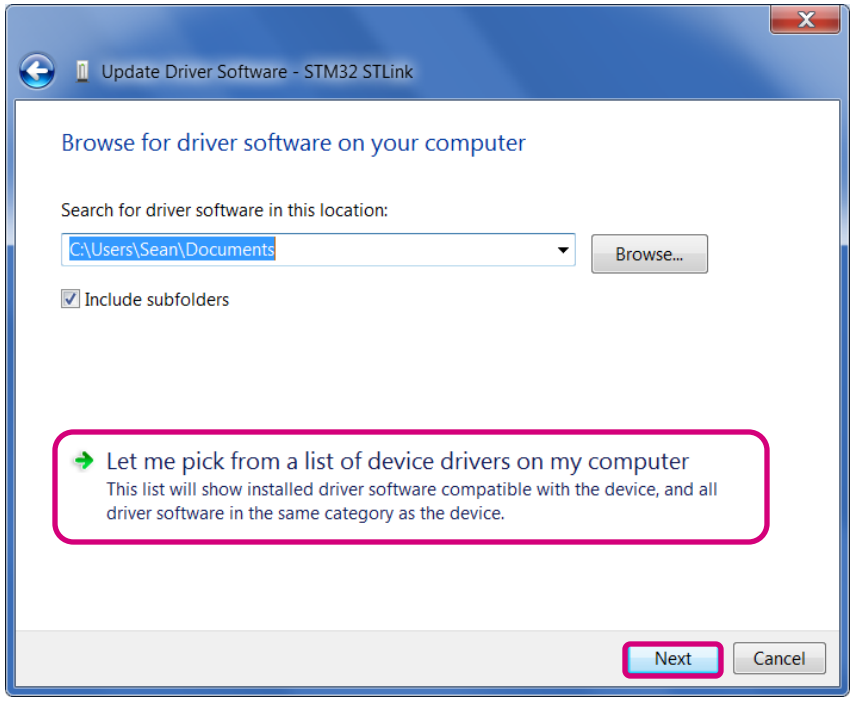
1. Open Device Manager
2. Right-click on the **STM32 STLink** Driver icon
3. Select “Update Driver Software”



Step #2 ST-Link Driver Trouble Shooting, (continued)



4. Select **"Browse my computer for driver software"**



5. Select **"Let me pick from a list of device drivers of my computer"**

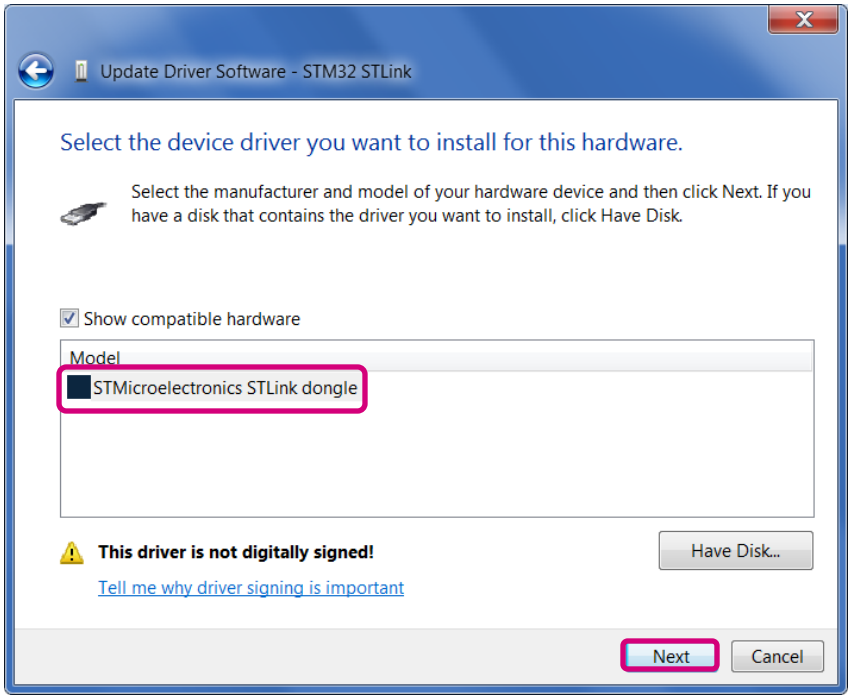
6. Click **"Next"**

life.augmented

Step #3 ST-Link Driver Trouble Shooting, (continued)

The **“STMicroelectronics ST-Link dongle”** should be listed

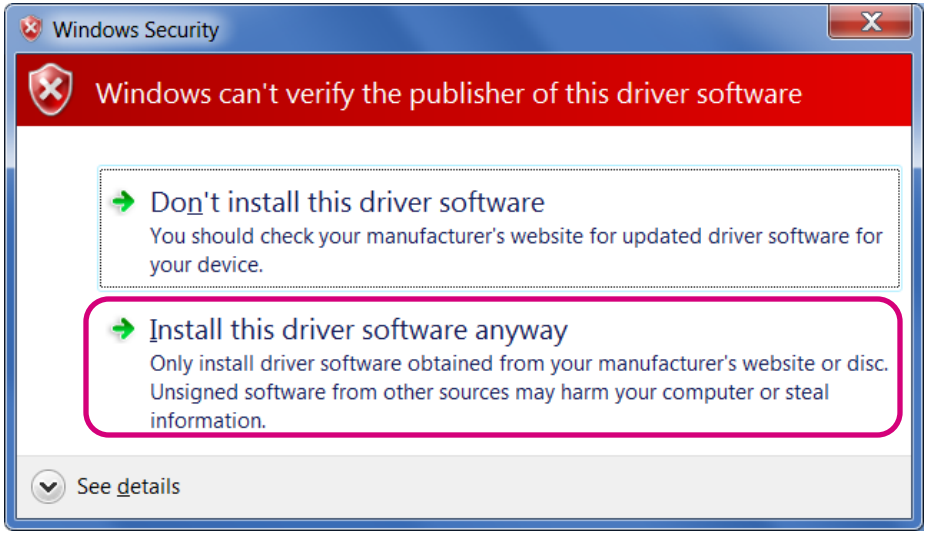
7. Click “Next”



Step #4 ST-Link Driver Trouble Shooting, (continued)

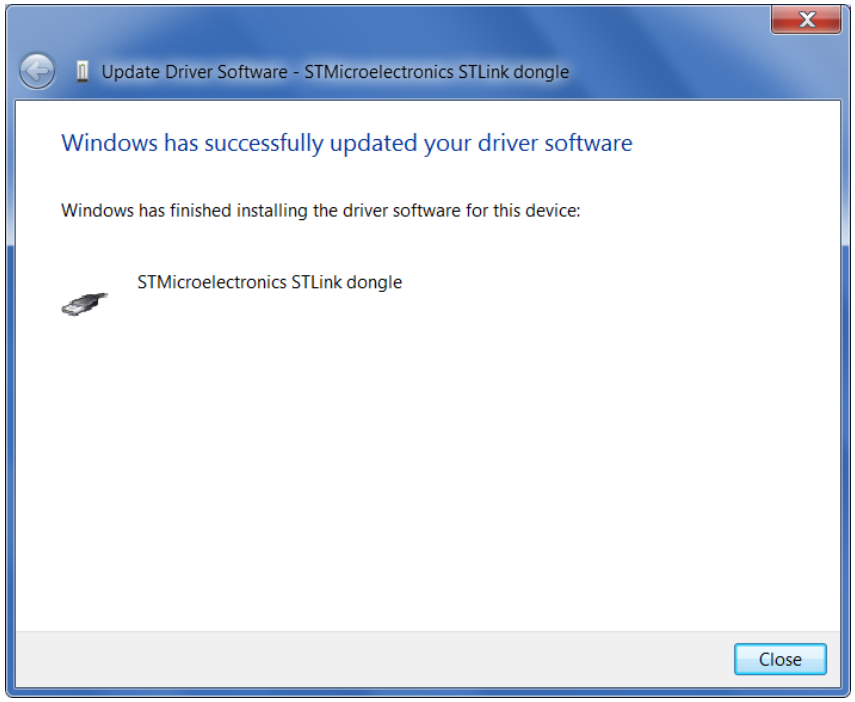
A warning message may appear

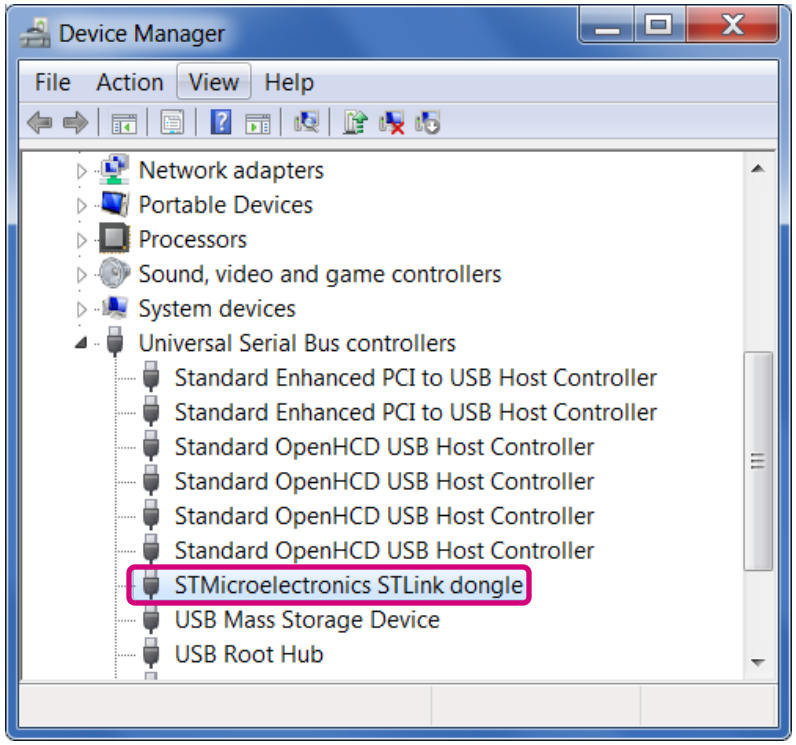
8. Select **“Install this driver software anyway”**



Step #5 ST-Link Driver Trouble Shooting, (continued)

- You should receive a message: **“Windows has successfully updated your driver software”**





- Re-check device manager to ensure “**STMicroelectronics ST-Link dongle”** is functioning normally

ST-LINK/V2
Windows® 8 Driver Installation Troubleshooting

life.augmented

Install ST-Link Driver (Alternate Method)

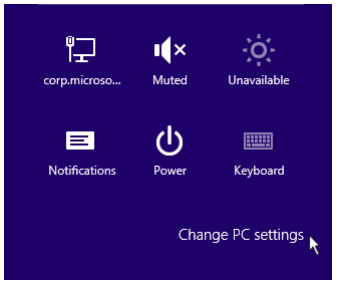
- The STM32F429 Discovery board includes an ST-Link/V2 embedded programming and debug tool
- The **older driver** for the ST-Link can be found at
  - C:\STM32Seminar\Utilities\STM32 ST-LINK-V2 Driver\ST-Link_V2_USBdriver.exe
- The Discovery board should be unplugged
- Double click on the file: ST-Link_V2_USBdriver.exe and click through the installation menu until the driver installation is complete.

Windows 8 Information

- This older ST-LINK/V2 Windows device driver is not certified for Windows 8
- In order to install this “unsigned” driver, the following steps should be followed in Windows 8
- These steps have been collected from the following third-party link:
  - http://www.trickday.com/2012/09/how-to-install-unsigned-drivers-in-windows-8/

Windows 8 – Specific Steps

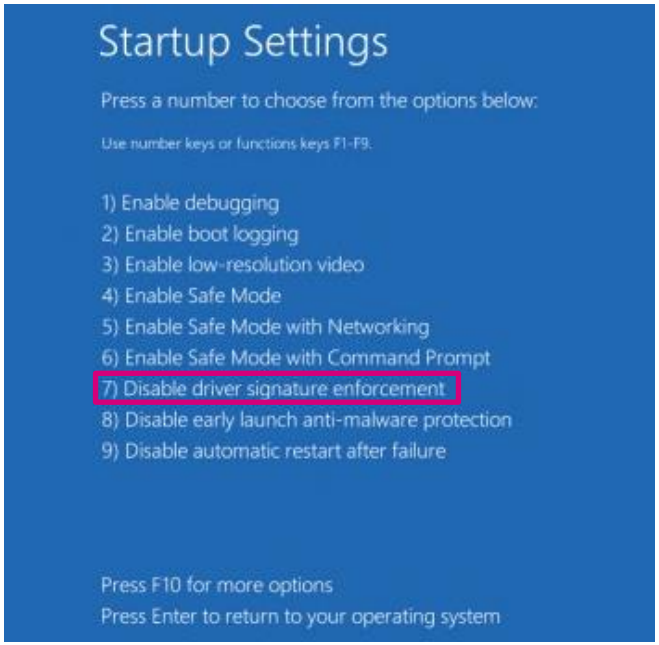
- Make sure you are signed in
- Press Windows+I to load the Charms Bar
  - Or, swipe from right if your device does not have a keyboard



- Click on “Power”, then hold down the Shift key, and click on “Restart” to restart the PC
- You should see a troubleshooting page come up. Click on “Troubleshoot here”, and then on the next page on “Advanced Options”.
- On the following screen that shows several options, click on “Startup Settings”

Windows 8 – Startup Settings

- On this screen, press 7 or F7 to “Disable driver signature enforcement”



- Windows 8 will restart automatically and the drivers you have installed should work from that moment on.

- While the preceeding slides may have fixed the issue, the problem may still persist for others. If this is the case, do the following:

- Restart Windows 8 in Advanced Startup Options Mode per previous slides. Once restarted in Advanced mode with driver signature enforcement disabled:

1) Uninstall any previously installed ST-Link/V2 driver

2) Reinstall ST-Link/V2 driver (ST-Link_V2_USBdriver.exe) that you had installed in the first step (do not let Windows search the web)

3) After driver installation go into Windows Device Manager and you may see the device “STM32 STLink” noted with or without a “!” indicating a problem. For both cases, right click the STM32 STLink and choose “Update Driver” and navigate to USB devices. You will find the ST-Link dongle in the list. Select this item and choose “ok”

- Now the ST-Link/V2 should be functional

Thumbdrive – What's there?

- Firmware
  - Specific Hands on Examples for today's workshop
  - Additional Project Examples specific to the F429 Discovery board
  - STM32F4xx Standard Peripheral Library, CMSIS and USB Host/Device/OTG Libraries
- Documentation
  - Board Schematic
  - Microcontroller, Gyro, LCD, SDRAM Datasheets
  - Application Notes for STemWIN, FPU, Upgrading board firmware via USB
  - Reference Manual
  - Programming Manual
  - Errata
- Additional Tools
  - Graphics related, MicroXplorer, IAR EWARM v6.60.2

Process check

- ST-Link is recognized by your system
- LD1 and LD2 (at top edge of board) should be ON and solid RED (indicating board power available and ST-Link is functional)
- LD3 and LD4 should be blinking in this demo
- LCD should display an interactive demo that you can try by touching the on-screen icons

Compile, Debug and Run

life.augmented

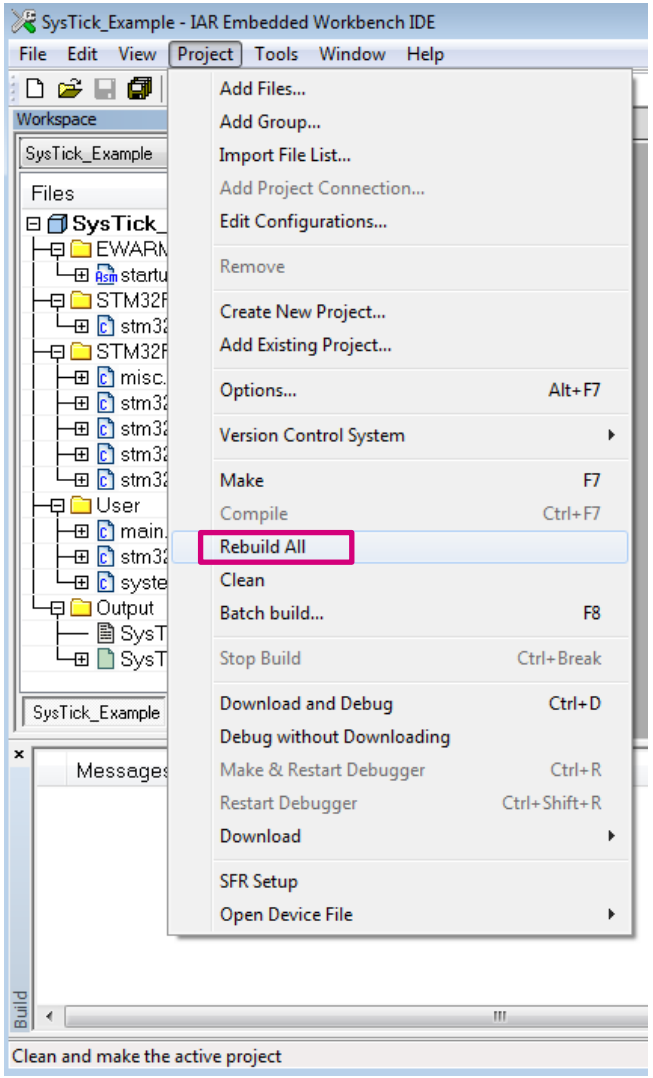
IAR SYSTEMS

Open “SysTick” Example Project with IAR EWARM

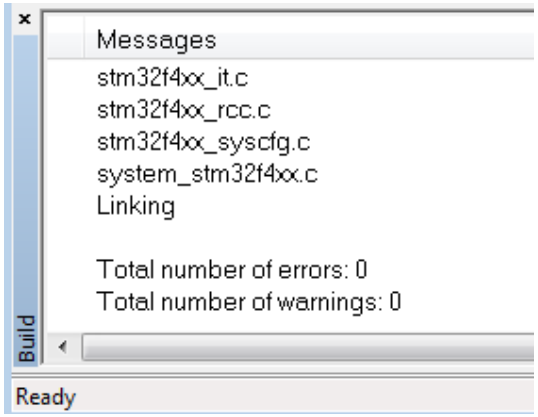
- Using explorer, go to the directory:
  C:\STM32Seminar\STM32F429I-Discovery_FW_V1.0.0\Projects\Peripheral_Examples\SysTick_Example\EWARM\
- Double-click on the SysTick_Example.eww file
- Alternatively, you can open this workspace from within IAR EWARM
  - File > Open > Workspace

Compile

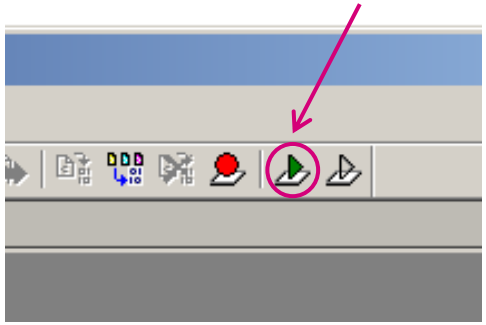
- Click on Menu > Project > Rebuild All



- The project should compile without errors:

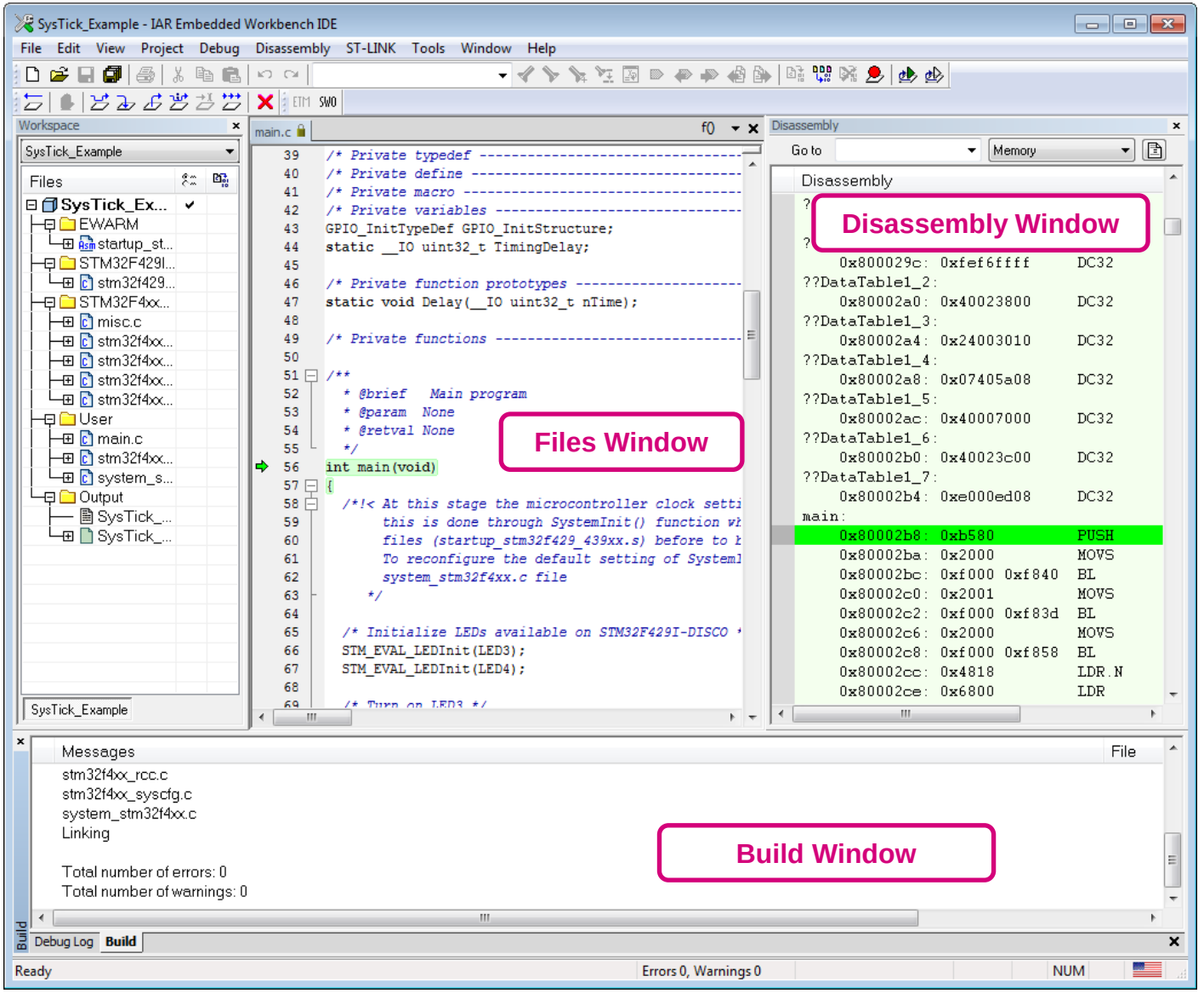


- Click on Menu > Project > Download and Debug or the Download and Debug button



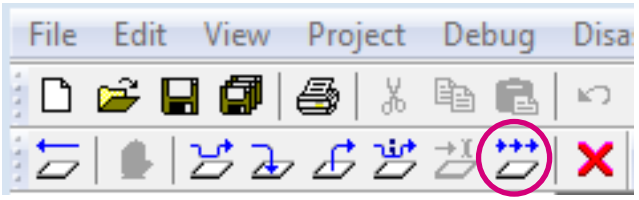
life.augmented

The C-SPY Debugger



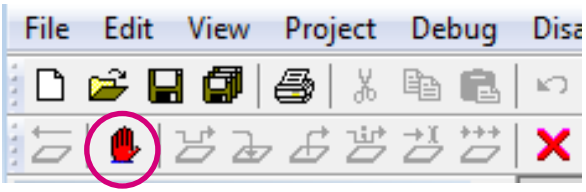
Run

- Click on the Go button to start the program
  - Or, you can use Menu > Debug > Go (or F5)

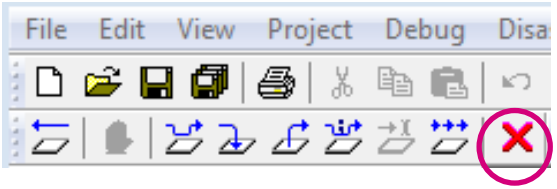


- Your STM32F429 DISCOVERY board LEDs LD3 and LD4 should blink alternately
  - LD1 (ST-Link Status) Should be also flashing

- Mission Accomplished
- Please click on the Break button.



- You code will stop anywhere within the program flow
- Click on the Stop Debugging button to exit from the debugger



Let’s make a change

- Double-click to open the main.c file
- Scroll down to line 110

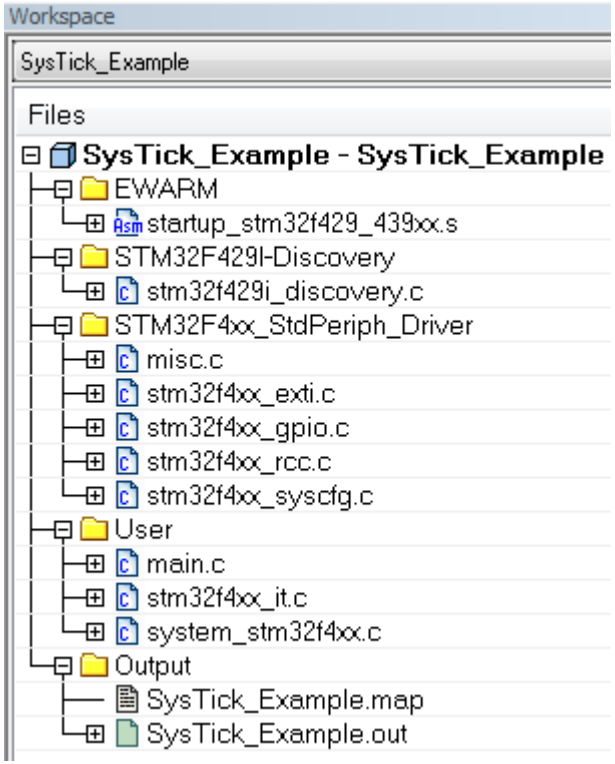
```
104   while (1)
105   {
106     /* Toggle LED4 */
107     STM_EVAL_LEDToggle(LED4);
108
109     /* Insert 50 ms delay */
110     Delay(50);
111
112     /* Toggle LED3 */
113     STM_EVAL_LEDToggle(LED3);
114
115     /* Insert 100 ms delay */
116     Delay(100);
117   }
```

- Modify the delay value from 50 to 1000 and place in the Delay(xxx) statement (this is in milliseconds)
- Compile, Debug, and Run
- Validate! Did it work?
- Stop debug and exit the debugger

Firmware Project Overview

Project Files

- User files
  - main.c (program entry point)
  - system_stm32f4xx.c (initial system configuration)
  - stm32f4xx_it.c (ISR's)
- stm32f429i_discovery.c
  - Board specific functions
- STM32F4xx_StdPeriph_Driver
  - Contains peripheral library functions
- startup_stm32f429_439xx.s
  - System initialization, vector table, reset and branch to main()



startup_stm32f429_439xx.s

- Main Characteristics
  - Initializes stack pointer

```
;* Description       : STM32F429xx/439xx devices vector table for EWARM toolchain.
;*                     This module performs:
;*                     - Set the initial SP
;*                     - Set the initial PC == _iar_program_start,
;*                     - Set the vector table entries with the exceptions ISR
;*                       address.
;*                     - Configure the system clock and the external SRAM/SDRAM mounted
;*                       on STM324x9I-EVAL board to be used as data memory (optional,
;*                       to be enabled by user)
;*                     - Branches to main in the C library (which eventually
;*                       calls main()).
;*                     After Reset the Cortex-M4 processor is in Thread mode,
;*                     priority is Privileged, and the Stack is set to Main.
;*******************************************************************************
```

  - Contains the vector table for the part

```
        DATA
__vector_table
        DCD     sfe(CSTACK)
        DCD     Reset_Handler             ; Reset Handler

        DCD     NMI_Handler               ; NMI Handler
        DCD     HardFault_Handler         ; Hard Fault Handler
        DCD     MemManage_Handler         ; MPU Fault Handler
        DCD     BusFault_Handler          ; Bus Fault Handler
        DCD     UsageFault_Handler        ; Usage Fault Handler
```

  - Contains Reset handler
    - This is called on system reset
    - Calls SystemInit() function
    - Calls __iar_program_start
      - Branch to main()

```
;;------------------------------------------------------------------------------
;; Default interrupt handlers.
;;
        THUMB
        PUBWEAK Reset_Handler
        SECTION .text:CODE:REORDER(2)
Reset_Handler

        LDR     R0, =SystemInit
        BLX     R0
        LDR     R0, =__iar_program_start
        BX      R0
```

system_stm32f4xx.c

- SystemInit()
  - This function is called at startup just after reset and before branch to main program. This call is made inside the "startup_stm32f429_439xx.s" file.
  - Sets up the system clock (System clock source, PLL Multiplier and Divider factors, AHB/APBx prescalers and Flash settings)

SystemInit()

```
187  void SystemInit(void)
188  {
189    /* FPU settings ------------------------------------------------------------*/
190    #if (__FPU_PRESENT == 1) && (__FPU_USED == 1)
191      SCB->CPACR |= ((3UL << 10*2)|(3UL << 11*2));  /* set CP10 and CP11 Full Access */
192    #endif
193    /* Reset the RCC clock configuration to the default reset state ------------*/
194    /* Set HSION bit */
195    RCC->CR |= (uint32_t)0x00000001;
```

Call SetSysClock()

```
216    /* Configure the System clock source, PLL Multiplier and Divider factors,
217       AHB/APBx prescalers and Flash settings ----------------------------------*/
218    SetSysClock();
```

Configure clock tree

```
320  static void SetSysClock(void)
321  {
322  /******************************************************************************/
323  /*            PLL (clocked by HSE) used as System clock source                */
324  /******************************************************************************/
325    __IO uint32_t StartUpCounter = 0, HSEStatus = 0;
326
327
328  #ifdef PLL_SOURCE_HSI
329
330        /* Configure the main PLL */
331      RCC->PLLCFGR = PLL_M | (PLL_N << 6) | (((PLL_P >> 1) -1) << 16) |
332                     (RCC_PLLCFGR_PLLSRC_HSI) | (PLL_Q << 24);
333
334  #else /* PLL_SOURCE_HSE_BYPASS or PLL_SOURCE_HSE */
```

main.c

- Example main()
  - Standard C main() function entry
  - Start of application program

```
56  int main(void)
57  {
58    /*!< At this stage the microcontroller clock setting is already configured,
59         this is done through SystemInit() function which is called from startup
60         files (startup_stm32f429_439xx.s) before to branch to application main.
61         To reconfigure the default setting of SystemInit() function, refer to
62         system_stm32f4xx.c file
63       */
64
65    /* Initialize LEDs available on STM32F429I-DISCO */
66    STM_EVAL_LEDInit(LED3);
67    STM_EVAL_LEDInit(LED4);
68
69    /* Turn on LED3 */
70    STM_EVAL_LEDOn(LED3);
```

stm32f4xx_it.c

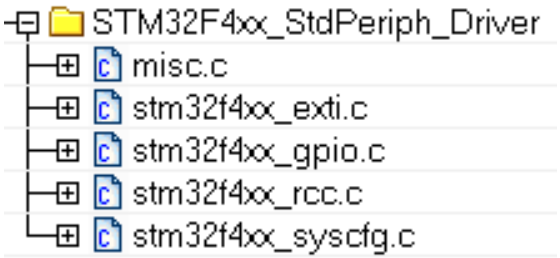
- Contains Cortex-M4 Processor Exception Handlers (ISRs)
  - void NMI_Handler(void);
  - void HardFault_Handler(void);
  - void MemManage_Handler(void);
  - void BusFault_Handler(void);
  - void UsageFault_Handler(void);
  - void SVC_Handler(void);
  - void DebugMon_Handler(void);
  - void PendSV_Handler(void);
  - void SysTick_Handler(void);
- Contains the STM32F4xx Peripherals Interrupt Handlers (default is empty)
- The Interrupt Handlers for the used peripheral(s) (PPP) can be added
  - For the available peripheral interrupt handler names please refer to the startup file: startup_stm32f429_439xx.s
  - Functions are added as follows (this overrides the weak definitions from the startup file): void PPP_IRQHandler(void) {};

stm32f429i_discovery.c

- Contains board specific functions and definitions
- Contains board specific functions to use the LEDs and button
  - void STM_EVAL_LEDInit(Led_TypeDef Led)
  - void STM_EVAL_LEDOn(Led_TypeDef Led)
  - void STM_EVAL_LEDOff(Led_TypeDef Led)
  - void STM_EVAL_LEDToggle(Led_TypeDef Led)
  - void STM_EVAL_PBInit(Button_TypeDef Button, ButtonMode_TypeDef Button_Mode)
  - uint32_t STM_EVAL_PBGetState(Button_TypeDef Button)

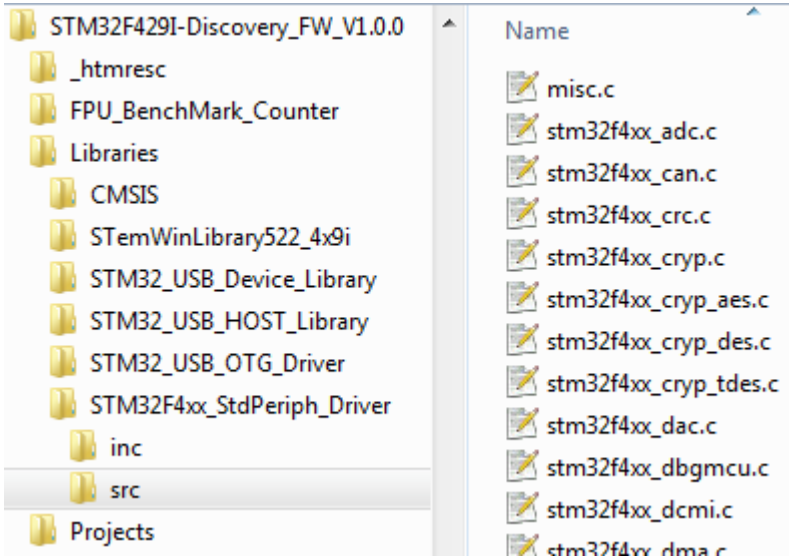
STM32F4xx_StdPeriph_Driver

- Each file contains library functions that can be used for each peripheral
- Abstracts register manipulation and gives a standard API for access to peripheral functions
- Example:



Only a subset is shown in project tree above, more peripherals can be added from the Libraries directory.

```
400 /**
401   * @brief  Sets the selected data port bits.
402   * @note   This functions uses GPIOx_BSRR register to allow atomic read/modify
403   *         accesses. In this way, there is no risk of an IRQ occurring between
404   *         the read and the modify access.
405   * @param  GPIOx: where x can be (A..K) to select the GPIO peripheral for STM3
406   *                      x can be (A..I) to select the GPIO peripheral for STM3
407   *                      x can be (A, B, C, D and H) to select the GPIO periphe
408   * @param  GPIO_Pin: specifies the port bits to be written.
409   *          This parameter can be any combination of GPIO_Pin_x where x can be
410   * @retval None
411   */
412 void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
413 {
414   /* Check the parameters */
415   assert_param(IS_GPIO_ALL_PERIPH(GPIOx));
416   assert_param(IS_GPIO_PIN(GPIO_Pin));
417
418   GPIOx->BSRRL = GPIO_Pin;
419 }
```



STM32F4 Graphics Workshop
Hands-on examples

life.augmented

CONTENTS

General guidelines

- You will need to update the example number in “main.c” for each exercise

  // !! TO BE MODIFIED !!

  #define EXAMPLE 1

- If you are unsure of where the code needs to be updated, you can search for “**// !! TO BE MODIFIED !!**”

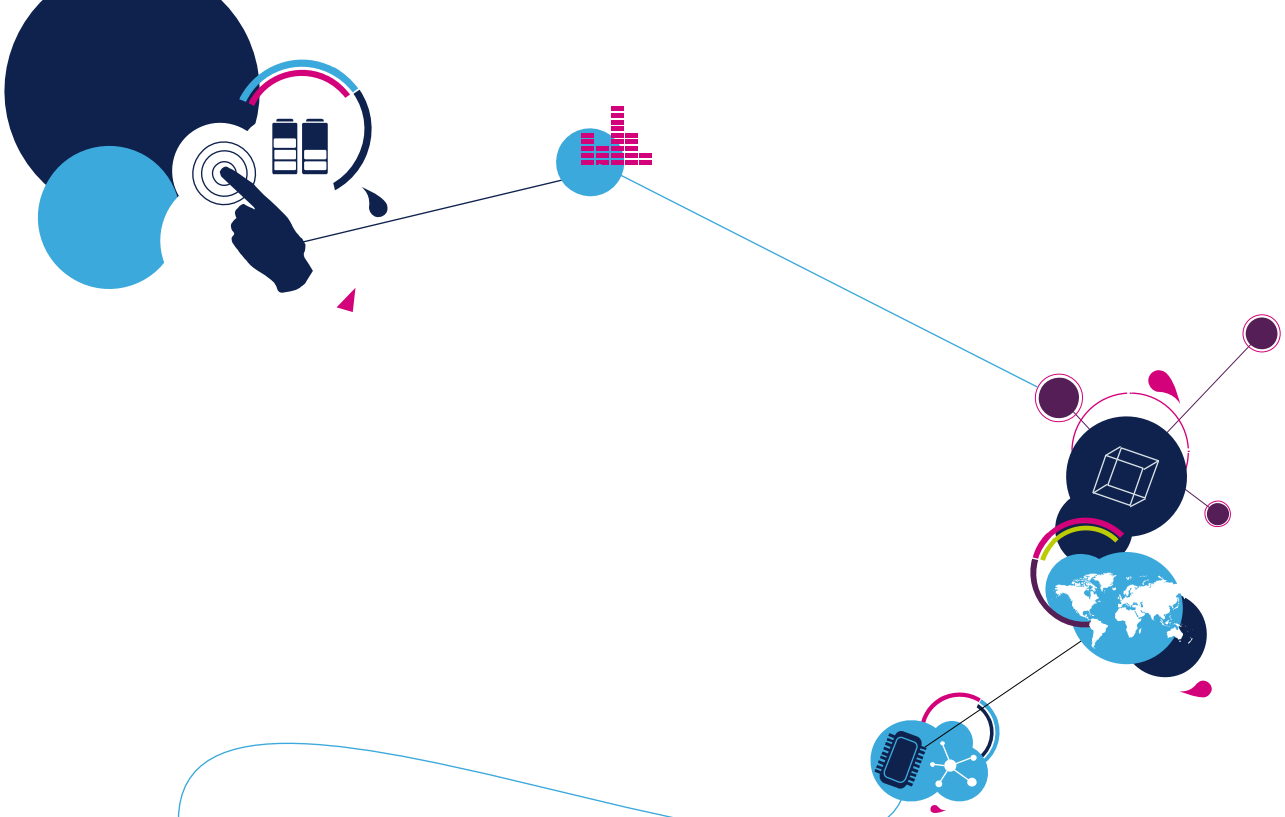
- The folder where all the example pictures are located is:

“…\Firmware\Projects\Demonstration\Media\”

- The folder where all the source file we will create should be saved to is:

“…\Firmware\Projects\Demonstration\Core\User_HandsOn\”

LTDC



life.augmented

1 Example Layer Scroll (1)

- Objective
  - Learn how to work with the LTDC Layer parameters
  - Learn how to convert a picture to C source code
- Tasks
  - Use the Scrolling layer example and change the picture to be scrolled
    - You can use a picture located in the example directory
    - Or, you can select your own picture larger than 320 x 240 (to see the scrolling effect) and raw size < 1MB (to fit into the internal flash memory)

1 Example Layer Scroll (2)

Prepare the picture to be used for scrolling:

- Open the image conversion tool located in
  C:\STM32Seminar\Tools\STM32 Imager.exe
- Open **any picture** you want to use (larger than 240x320 pixels, smaller than 1MB in raw size)
  - Available example: c:\stm32seminar\Firmware\Projects\Demonstration\Media\kristynka.jpg
- Set “C Table Name” as:
  Image
- Set Output color format as:
  24bpp (RGB888)
- Export the picture
  - Store the file in the project directory (overwriting the original picture):

C:\stm32seminar\Firmware\Projects\Demonstration\Core\User_HandsOn\image.h

1 Example Layer Scroll (3)

- Open the following workspace with IAR EWARM
  - C:\stm32seminar\Firmware\Projects\Demonstration\EWARM_HandsOn\STM32F429I-Discovery_Demo.eww
  - Open the “main.c” file
  - Check that you have the correct example enabled with the following define:

```
// !! TO BE MODIFIED !!
#define EXAMPLE 1
```

- Open the “main_layerscroll.c” file
  - Look for the occurrences of the “Image”, “Image_width” and “Image_height” keywords and modify the source code as necessary to use the new image
    - Be aware of the name of the variables and array declared in “image.h” when modifying “main_layerscroll.c” to use the new image
  - Explore the source code to see the steps necessary to display the picture
- Recompile and run …

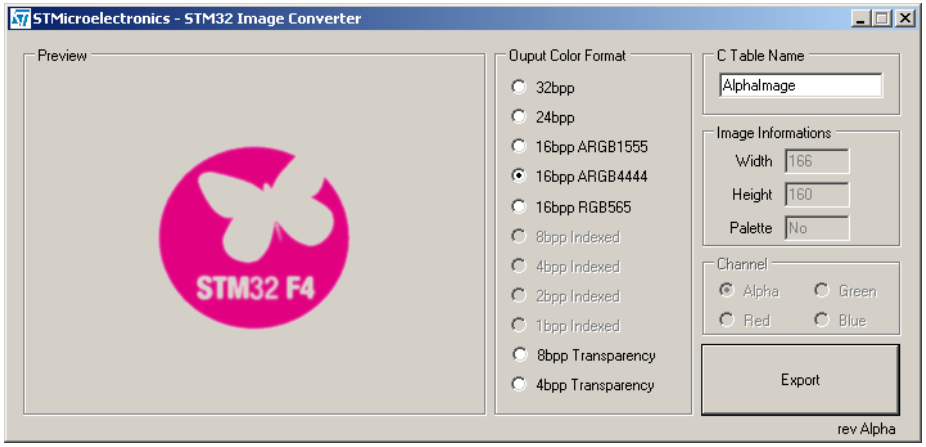
Example Alpha Blending (1)

- Objective
  - Learn how to work with alpha channels and layer blending
- Tasks
  - Take the Two Layers example and change the scrolling picture
    - You can select your own picture which already includes the alpha layer. The size of the picture should be smaller than 240 x 320 to see its movement
    - You can also use a picture located in the example directory
- Creation of a picture with an alpha layer
  - Working with the graphic SW is out of scope for this seminar
  - Recommended tutorials: http://www.youtube.com/watch?v=LQCziSTNJgQ
  - Search YOUTUBE.COM for "Gimp alpha channel tutorial“

Example Alpha Blending (2)

- Prepare the picture to be used for scrolling
- Open the image conversion tool located in
  - C:\stm32seminar\Tools\STM32 Imager.exe
- Open **any picture with an alpha channel** you want to use (size smaller than 240x320 pixels)
  - Available example: c:\stm32seminar\Firmware\Projects\Demonstration\Media\stm32f4.png
- Set “C Table Name” as:
  - i.e. **AlphaImage**
- Set Output color format as:
  - **16bpp ARGB4444**
- Export the picture
  - Store the result into the project directory:

C:\stm32seminar\Firmware\Projects\Demonstration\Core\User_HandsOn\alphaimage.h



Example Alpha Blending (3)

- Open the following workspace with IAR EWARM
  - C:\stm32seminar\Firmware\Projects\Demonstration\EWARM_HandsOn\STM32F429I-Discovery_Demo.eww
  - Open the “main.c” file
  - Modify example to number **2** the same way as for example 1:

  // !! TO BE MODIFIED !!

  #define EXAMPLE **2**

- Open the “main_2layers.c” file
- Include your newly created header file into “main_2layers.c”

  Ex: #include “alphaimage.h” replaces #include “ImageSTLogoTransparent.h”

- Modify the LCD_LogoLayer() function to use the new image
  - Replace “ImageSTLogoTransparent…” with “AlphaImage…”
  - Check that the proper color format selection is used in the **LTDC_PixelFormat** init structure member
  - Set the correct multiplication factor for “bytes per pixel” for the following init structure members:
    - LTDC_CFBLineLength
    - LTDC_CFBPitch
- Modify the LogoLayerScrollVBCallback() function to use the image size attributes
  - Replace “ImageSTLogoTransparent…” with “AlphaImage…”
- Recompile and run …

Example GUI Creation (1)

- Objective
  - Learn how to use the Chrome ART (DMA2D)
- Tasks
  - Take the GUI Content Creation example and add more icons

Example GUI Creation (2)

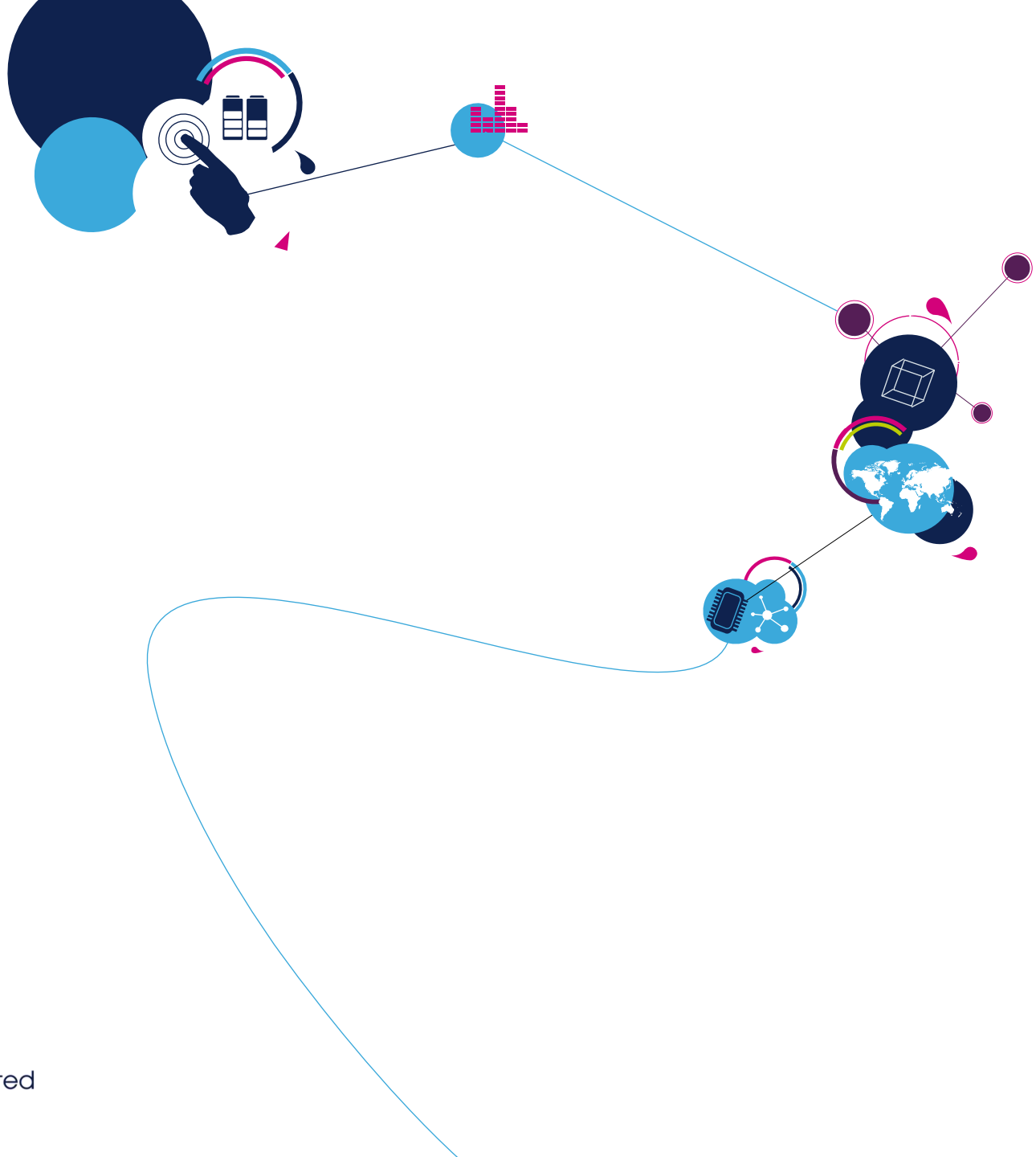
- Open the following workspace with IAR EWARM
  - C:\stm32seminar\Firmware\Projects\Demonstration\EWARM_HandsOn\STM32F429I-Discovery_Demo.eww
  - Open the “main.c file”
  - Modify example to number **3** the same way as before:

```
// !! TO BE MODIFIED !!
#define EXAMPLE 3
```

- Open the “main_contentCreation.c” file
- First, add an “ARMED” button that will be displayed
  - The part of the source code to be updated is marked as follows:

```
// !! TO BE MODIFIED !!
```

- Recompile and run …



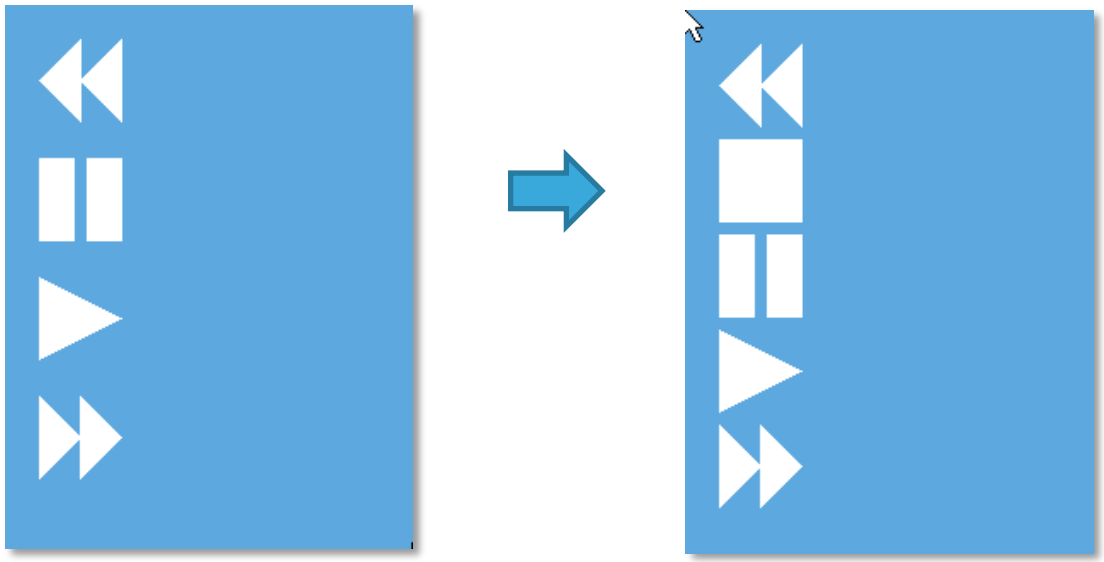
STemWin

Hands-On session

life.augmented

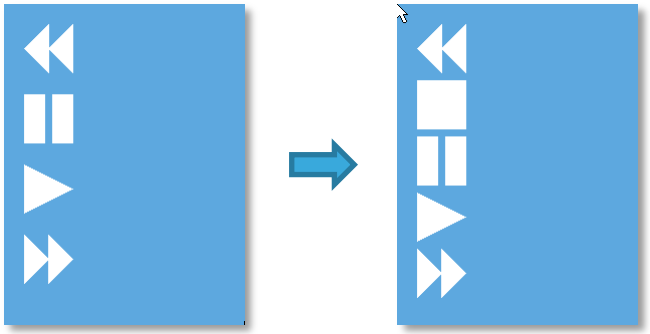
4 Example PLAYER application

- You will see how easy it is to change the looks of the GUI in STemWin
- There is a missing element which will be added at each step of the example PLAYER application
- Each part of the source code to be updated is marked as follows:
  `// !! TO BE MODIFIED !!`
- If necessary, use the STemWin library user manual:
  c:\STM32Seminar\Firmware\Libraries\STemWinLibrary522_4x9i\Documentation\STemWin5.pdf



4 Add STOP button

- Open the “main_2D.c” file
- Add the DrawStop() function into the iconDrawFunctions[] function list
- Change the content of DrawStop() to make a rectangle



- *You can change the color of the background, of all the elements, or of individual elements*

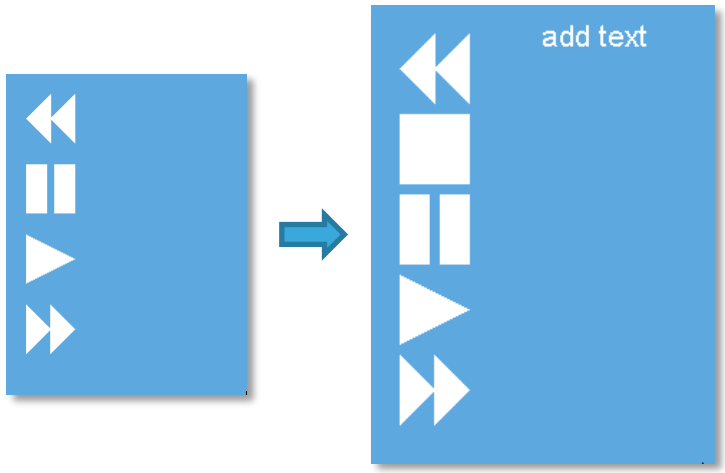
Examples:

```
GUI_SetBkColor(GUI_GREEN); //changes the background
color
GUI_SetColor(GUI_RED); //changes the color for the
next draw operations
```

life.augmented

Add text to the example

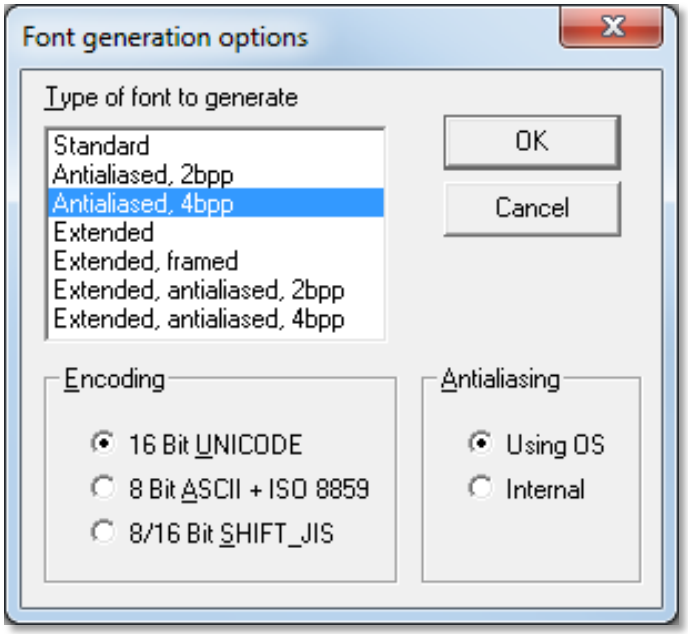
- Open the “main_2D_text.c” file
- Insert some text on the right side of the icons
- Select a font from built-in fonts (see p. 226 of the STemWin documentation for available fonts)
- *You can change the color of text*
  - Example:




```
GUI_SetFont(&GUI_Font24_1);
GUI_SetColor(GUI_WHITE);
GUI_DispStringInRectWrap(“add text", &rect, GUI_TA_CENTER,
GUI_WRAPMODE_WORD);
```

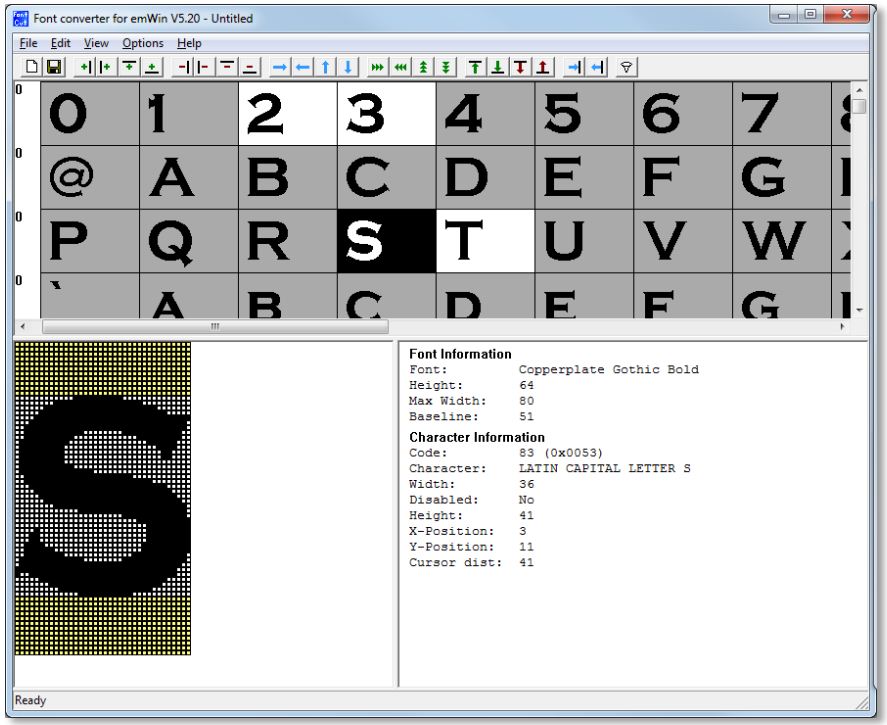
Create your own font

- You can use any Windows font to create a font with anti-aliasing
- Use the FontConverter
  - Install it from: “c:\stm32seminar\Tools\SetupFontCvt_V522.exe”, double click and click through the installation process
  - Start the application
  - Click OK



Create your own font

Disable all characters (Edit → Disable all) to save memory. Then, use the “space” key to select only the characters you want to use, and finally save everything as a .c file



- Add the generated .c file into the project and use it

```
GUI_SetFont(&GUI_FontTimesNewRoman31);
GUI_DispStringInRectWrap("STM32 Player", &rect, GUI_TA_CENTER,
GUI_WRAPMODE_WORD);
```

- Open the “main_2D_AAtext.c” file

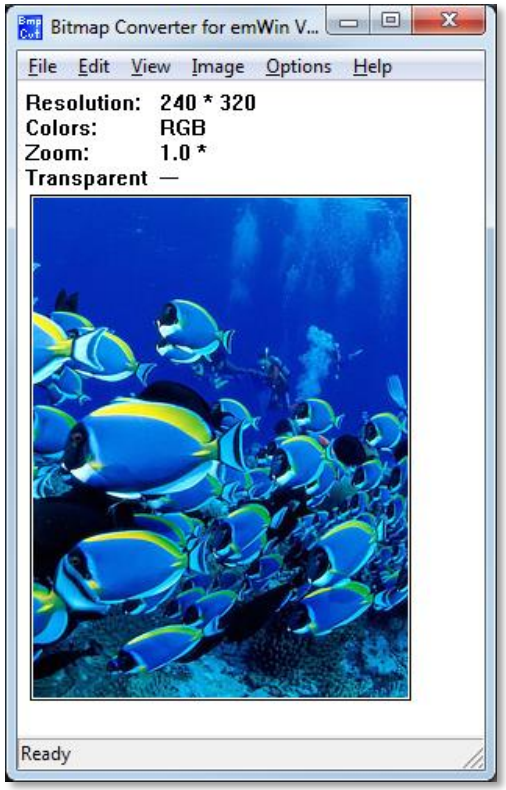
7 Insert a bitmap as background

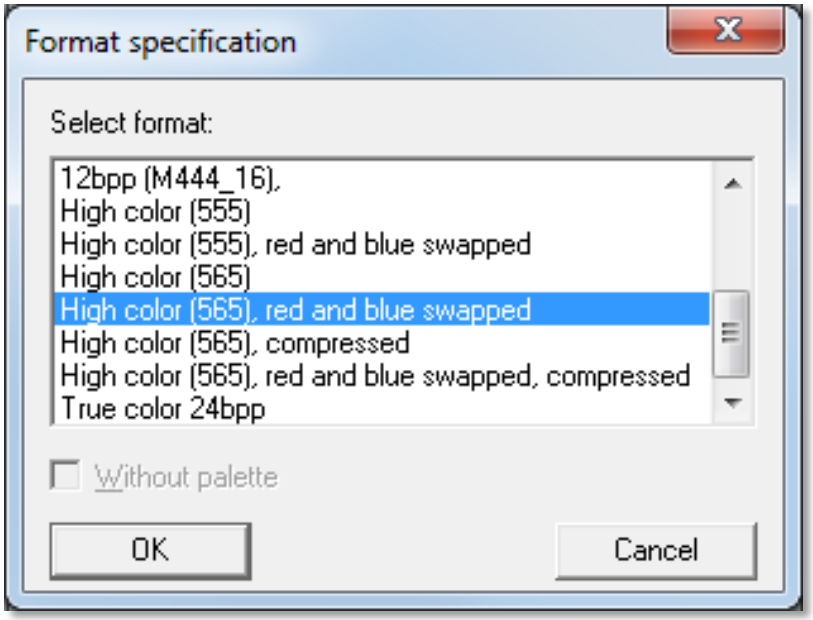
- Open the “main_2D_text_bmp.c” file
- Insert any picture as background. You can use the “background.bmp” picture provided or use any other 240*320 bitmap file.
- You can create your own bmp using the “BmpCvt” SW: “C:\stm32seminar\Tools\BmpCvt.exe”
- A bmp is imported and used as background in the code as follows:

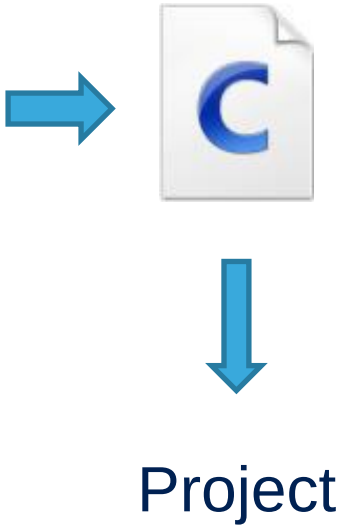
```
extern GUI_CONST_STORAGE GUI_BITMAP bmbackground;
GUI_DrawBitmap(&bmbackground, 0, 0);
```

7 Insert bitmap as background

- “File→Save as” the bitmap file as a .c file in RGB565 format with swapped Red and Blue





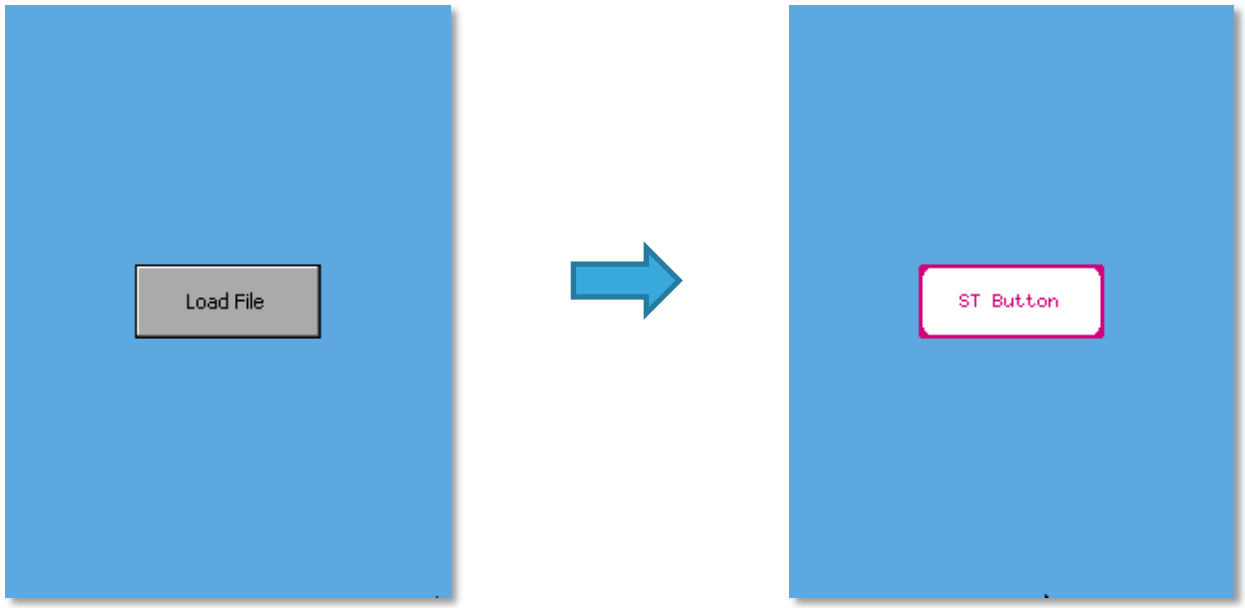


Project

Skinning of widgets

- Open the “main_WM_button_skin.c” file
- The widgets appearance can be changed by user defined skinning
- To do that, just change the drawing callback function of the button:

```
BUTTON_SetSkin(hButton, DrawSkinST_BUTTON);
```

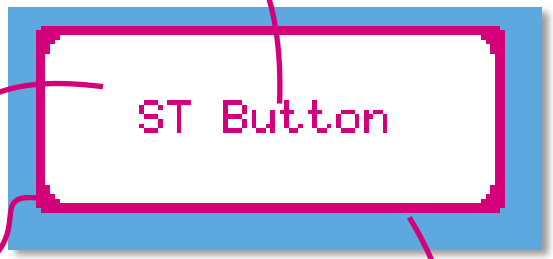


Skinning of widgets


```
static int DrawSkinST_BUTTON(const WIDGET_ITEM_DRAW_INFO * pDrawItemInfo) {
#define RAD 5
char acBuffer[20];
GUI_RECT Rect;
switch (pDrawItemInfo->Cmd) {
    case WIDGET_ITEM_DRAW_TEXT:
        BUTTON_GetText(pDrawItemInfo->hWin, acBuffer, sizeof(acBuffer));
        GUI_SetColor(STPINK);
        Rect.x0 = pDrawItemInfo->x0; // Default position of text
        Rect.y0 = pDrawItemInfo->y0;
        Rect.x1 = pDrawItemInfo->x1;
        Rect.y1 = pDrawItemInfo->y1;
        GUI_SetTextMode(GUI_TM_TRANS);
        GUI_DispStringInRect(acBuffer, &Rect, GUI_TA_VCENTER | GUI_TA_HCENTER);
        break;
    case WIDGET_ITEM_DRAW_BACKGROUND:
        GUI_SetPenSize(3);
        // draw background
        Rect.x0 = pDrawItemInfo->x0;
        Rect.y0 = pDrawItemInfo->y0;
        Rect.x1 = pDrawItemInfo->x1;
        Rect.y1 = pDrawItemInfo->y1;
        if (pDrawItemInfo->ItemIndex == BUTTON_SKINFLEX_PI_PRESSED)
           GUI_SetColor(GUI_GRAY);
        else
           GUI_SetColor(GUI_WHITE);
        GUI_FillRoundedRect(Rect.x0, Rect.y0, Rect.x1, Rect.y1, RAD);
        // draw outline
        GUI_SetColor(STPINK);
        GUI_DrawArc(Rect.x0 + RAD, Rect.y0 + RAD, RAD, RAD, 90, 180);
        GUI_DrawArc(Rect.x1 - RAD, Rect.y0 + RAD, RAD, RAD, 0, 90);
        GUI_DrawArc(Rect.x1 - RAD, Rect.y1 - RAD, RAD, RAD, 270, 359);
        GUI_DrawArc(Rect.x0 + RAD, Rect.y1 - RAD, RAD, RAD, 180, 270);
        GUI_DrawLine(Rect.x0 + RAD, Rect.y0, Rect.x1 - RAD, Rect.y0);
        GUI_DrawLine(Rect.x1, Rect.y0 + RAD, Rect.x1, Rect.y1 - RAD);
        GUI_DrawLine(Rect.x0 + RAD, Rect.y1, Rect.x1 - RAD, Rect.y1);
        GUI_DrawLine(Rect.x0, Rect.y0 + RAD, Rect.x0, Rect.y1 - RAD);
    break;
    Default:
    return BUTTON_DrawSkinFlex(pDrawItemInfo);
}
}
```

Enter text string

Background depends on pressed state



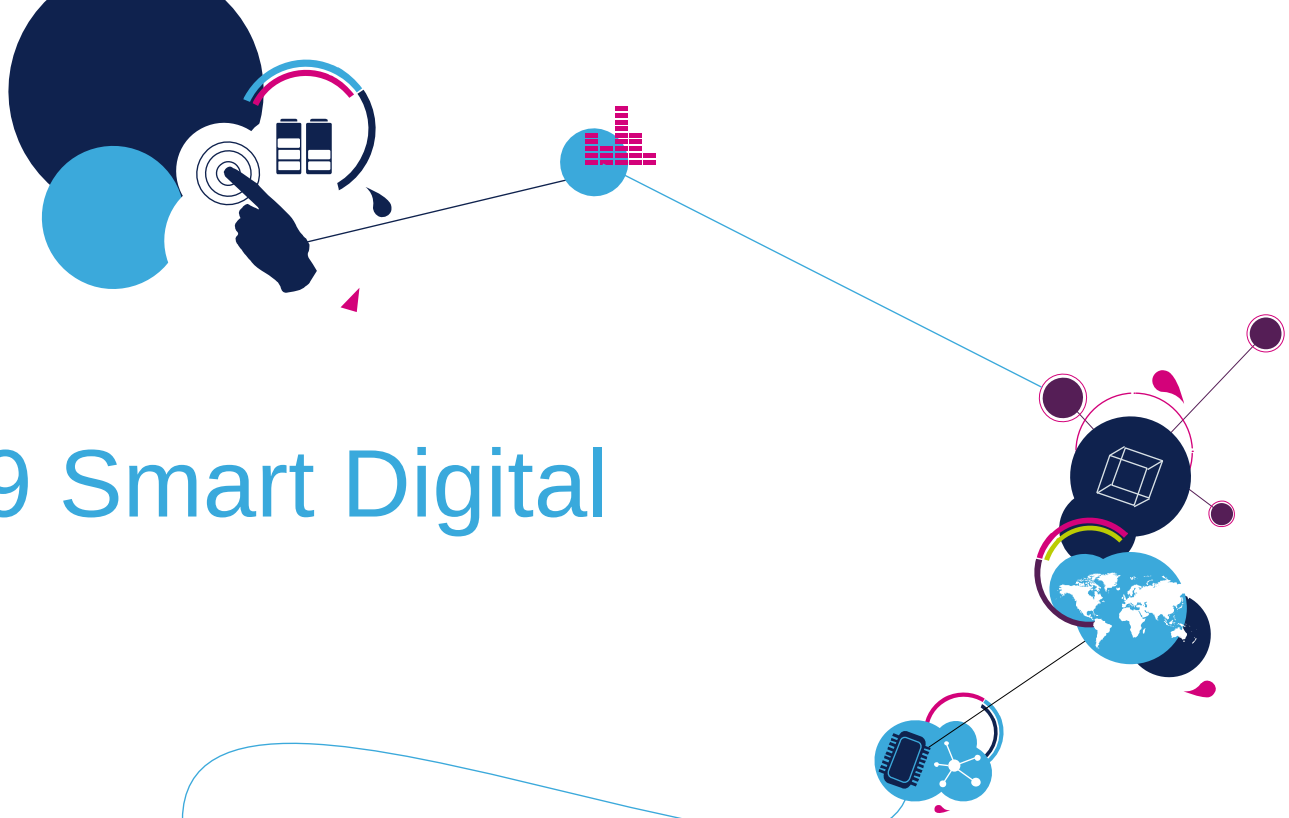
Round corners

Frame

life.augmented

www.st.com/stm32f4

life.augmented



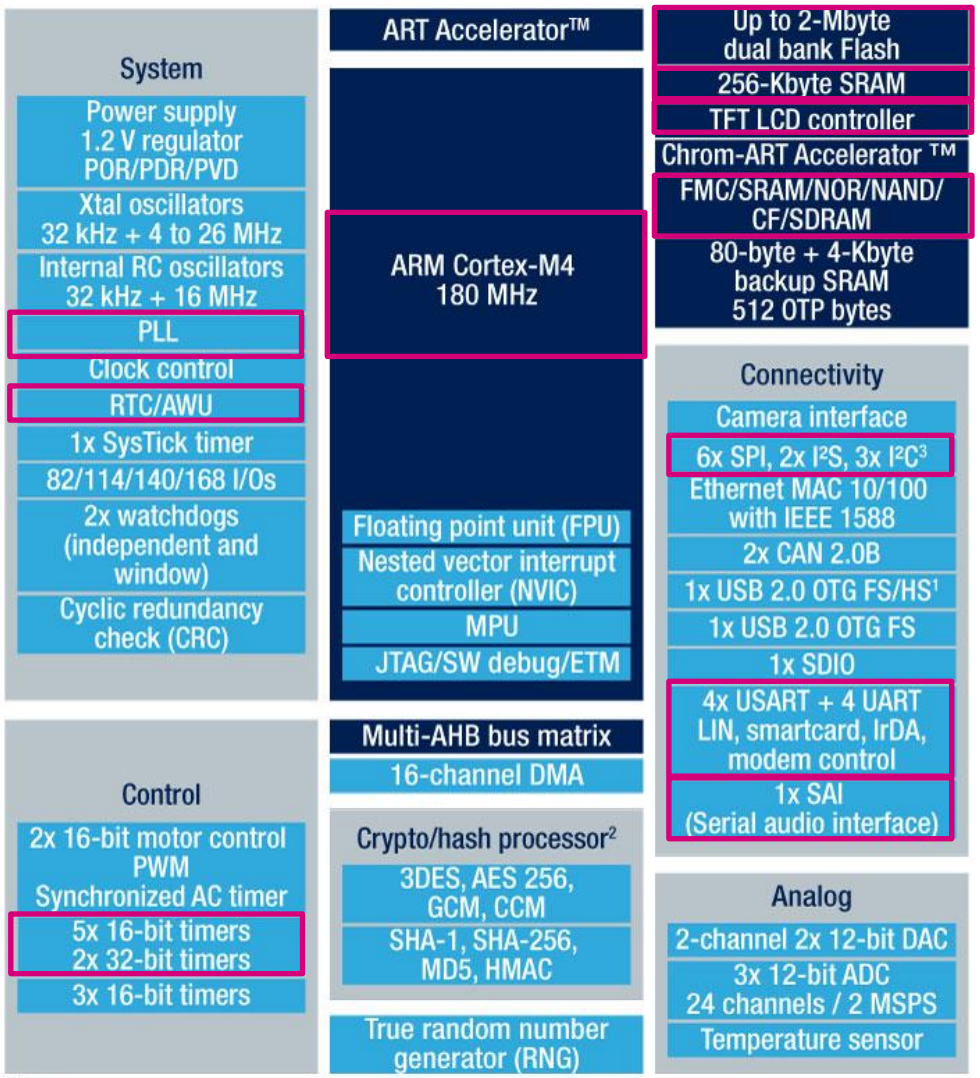
STM32F429 Smart Digital Peripherals

life.augmented

STM32 F4 2MB

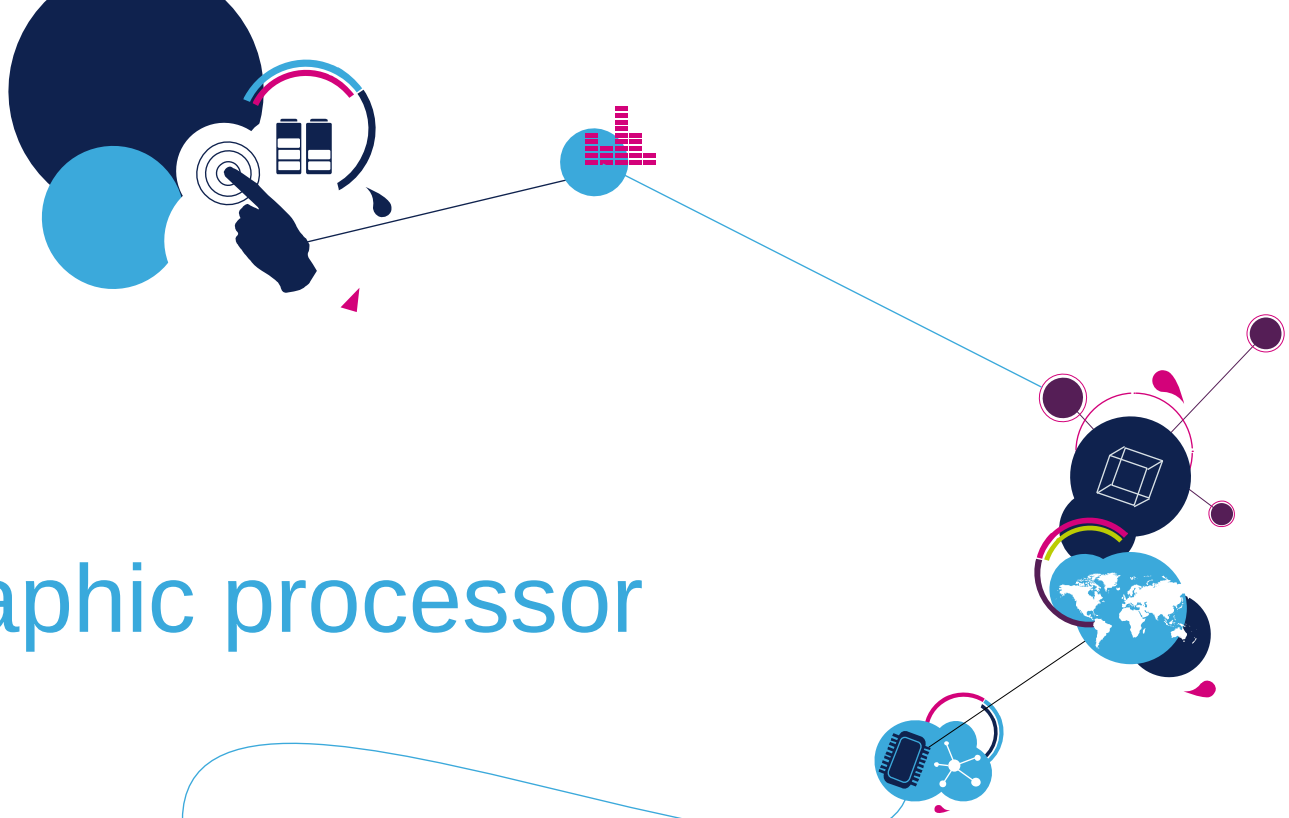
- Cortex-M4 w/FPU, 180 MHz
- Full compatible F2/F4
- Dual Bank 2 x 1MB Flash
- 256KB SRAM
- FMC with SDRAM Support, 32-bit data on 208-pin and 176 packages, 16-bit else
- Serial Audio I/F with PCM/TDM support
- TFT LCD Controller with DMA-2D
- Hash: supporting SHA-2 and GCM
- More serial com and more fast timers running at Fcpu
- 100- to 208-pin

New feature



Notes:
1. HS requires an external PHY connected to the ULPI interface
2. Crypto/hash processor on STM32F415, STM32F417, STM32F437 and STM32F439
3. With digital filter feature

life.augmented



Cryptographic processor

life.augmented

Definitions

- ***Plaintext*** is the original text/values to be encrypted; the original message.
- ***Encryption*** is the process used to convert plaintext into a form that is not readable without knowledge of the rules and key to decrypt.
- ***Decryption*** is the process used to convert from an encrypted form back to plaintext, using a key.
- A ***key*** is the value string used for both encryption and decryption.

Encryption/ Decryption Modes

- Electronic code book(ECB),
- Cypher Block Chaining(CBC) and
- Counter(CTR) modes.

Allows encryption only .

- **GCM: AES Galois/Counter Mode**
- **CCM: Combined cypher Machine**

allows encryption and authentication

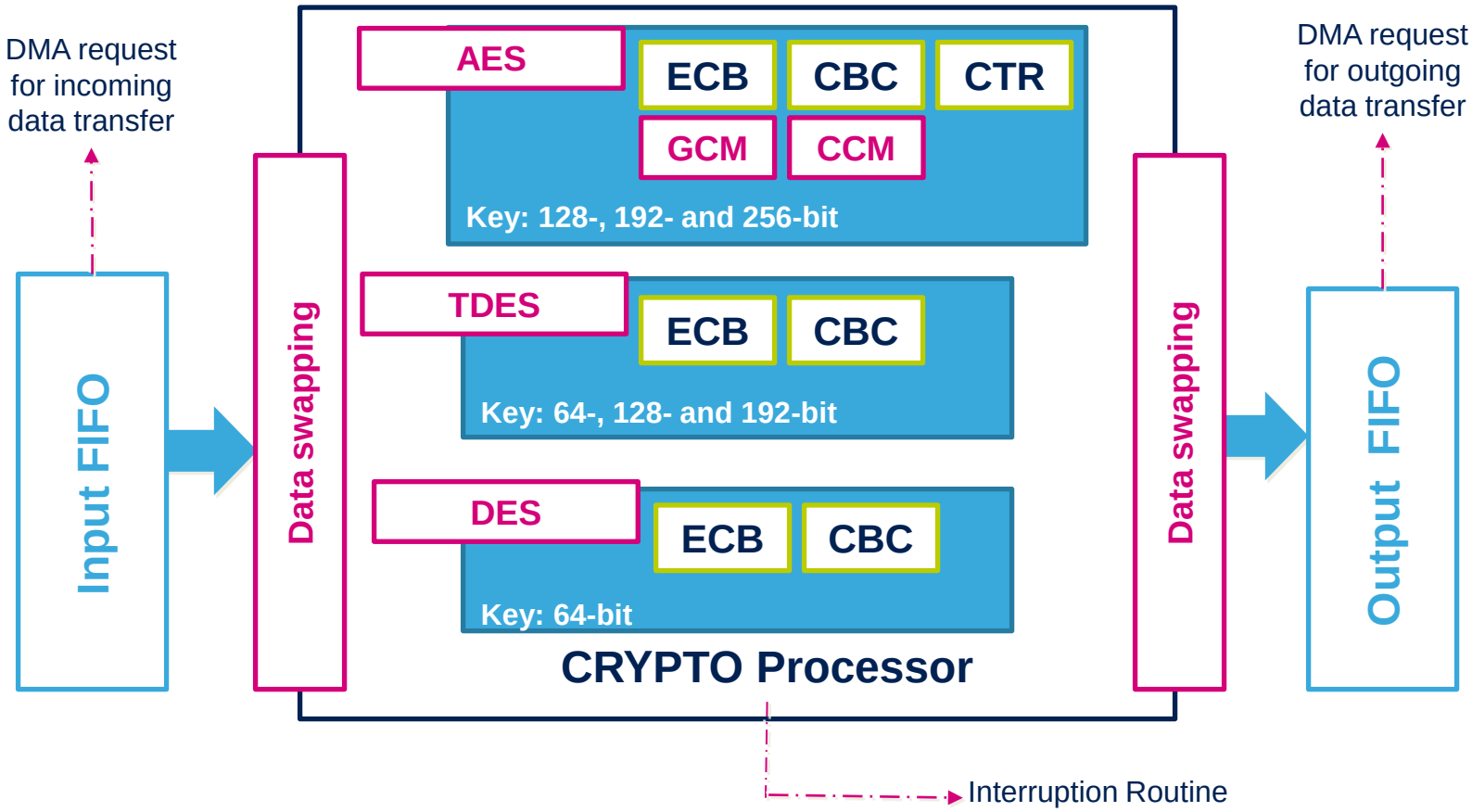
- Two other Modes are derived from the new modes:
  - GMAC: Galois Message Authentication Code
  - CMAC: cypher Message Authentication Code

allows authentication only

CRYP algorithms overview

	AES	DES	TDES
Key sizes	128, 192 or 256 bits	64* bits * 8 parity bits	192***, 128** or 64* bits * 8 parity bits : Keying option 1 ** 16 parity bits: Keying option 2 ***24 parity bits: Keying option 3
Block sizes	128 bits	64 bits	64 bits
Time to process one block	14 HCLK cycle for key = 128bits 16 HCLK cycle for key = 192bits 18 HCLK cycle for key = 256bits	16 HCLK cycles	48 HCLK cycles
Type	block cypher	block cypher	block cypher
Structure	Substitution-permutation network	Feistel network	Feistel network
First published	1998	1977 (standardized on January 1979)	1998 (ANS X9.52)

CRYP Block Diagram



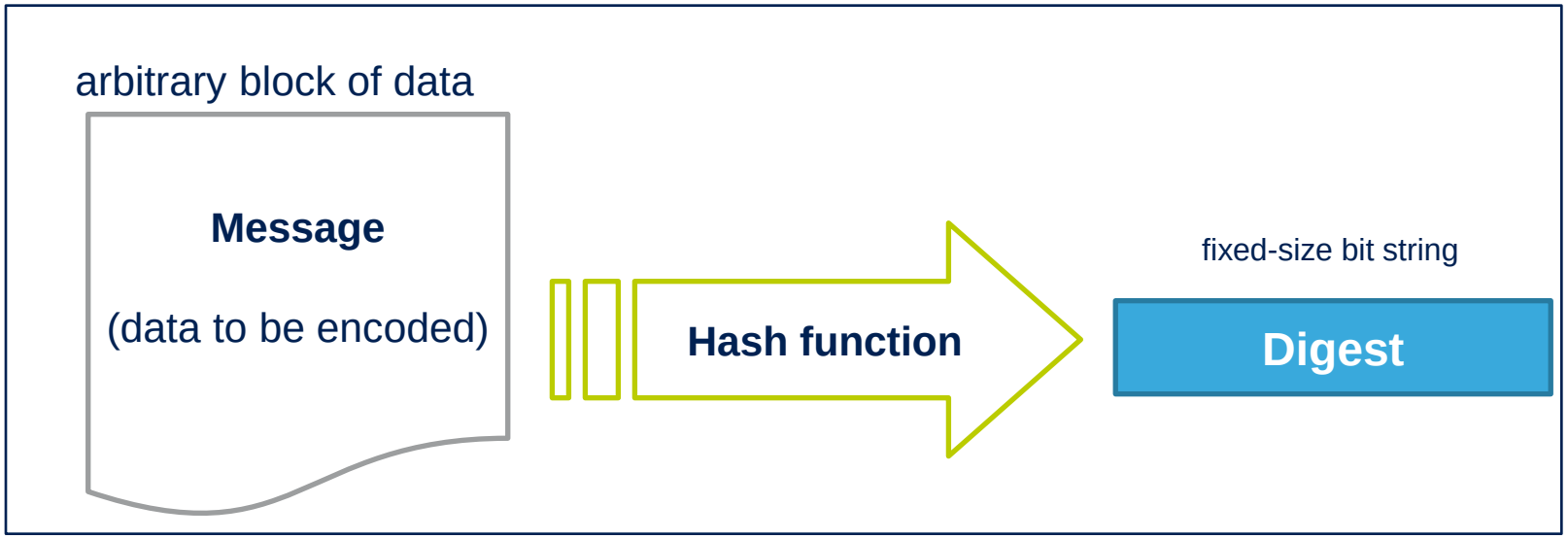
New Encryption/ Decryption Modes

Hash processor

life.augmented

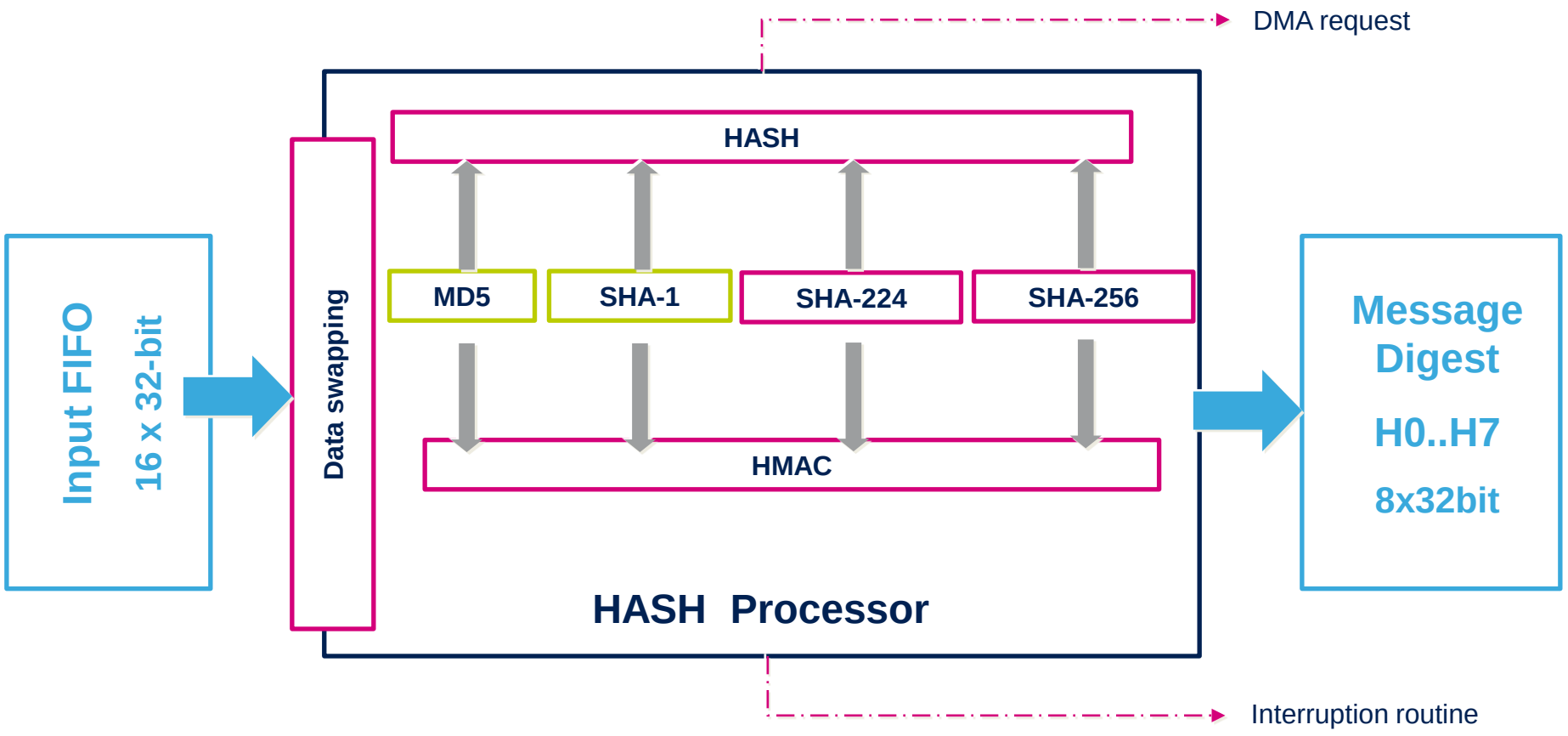
Definitions

- A **cryptographic hash function** is the transformation of a message into message digest with usually shorter fixed-length value that depend of Hash algorithm applied.



- MD5: message digest size is 128bits
- SHA-1: message digest size is 160 bits
- **SHA-224: message digest size is 224 bits**
- **SHA-256: message digest size is 256bits**

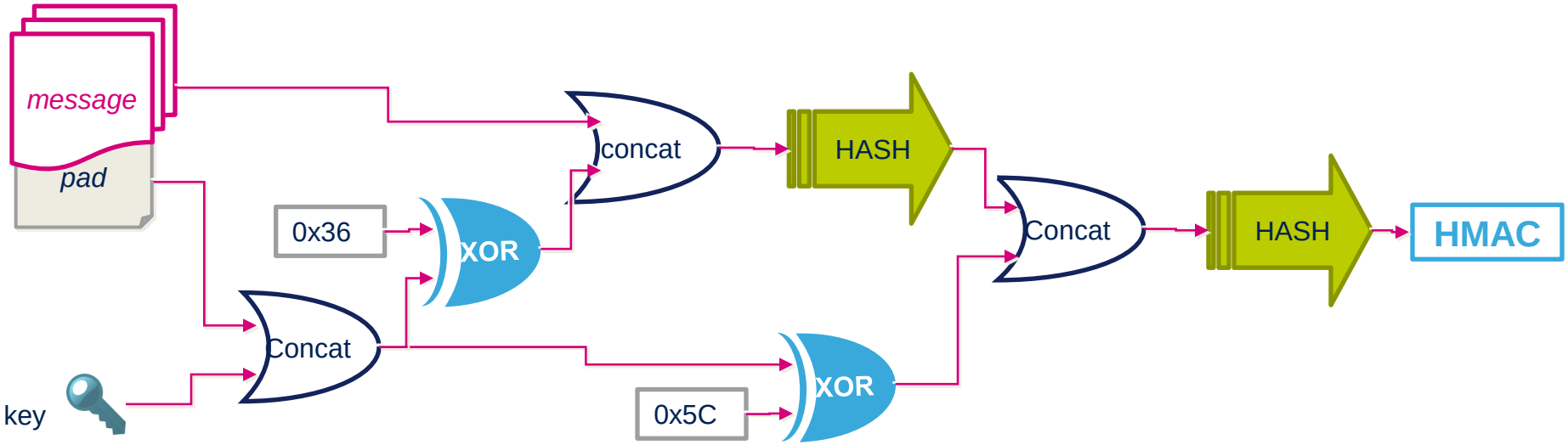
HASH Block Diagram



New Hash algorithms

HMAC operation

- The HMAC algorithm is used for message authentication, by irreversibly binding the message being processed to a key chosen by the user.
- For HMAC specifications, refer to “HMAC: keyed-hashing for message authentication, H. Krawczyk, M. Bellare, R. Canetti, February 1997.
- Basically, the algorithm consists of two nested hash operations:



SAI - Serial Audio Interface

life.augmented

SAI Features(1/3)

- Peripheral with large configurability and flexibility allowing to target the following audio protocol:
  - **I2S standards** (SAI = 2xI2S half duplex or 1xI2S full duplex),
  - **LSB or MSB-justified,**
  - **PCM** with “DSP” mode support
  - **TDM** (Time Division Multiplex) ➔ Allows multiple connections (DACs, Bluetooth chip…) over the bus
  - **AC’97** protocol
- Two independent audio sub-blocks which can be
  - Transmitter and/or receiver
  - Master or slave
  - Synchronous or asynchronous mode between the audio sub-blocks
  - Clock generator for each audio sub-block to target independent audio frequency sampling

SAI Features(2/3)

- **8-word integrated FIFOs** for each audio sub-block.
- LSB first or MSB first for data transfer.
- Mute mode
- Stereo/Mono audio frame capability.
- Communication clock strobing, edge configurable (SCK).
- Companding mode supported **µ-Law** and **A-Law**.
- **Up to 16 slots** available with configurable size and with the possibility to select which ones are active in the audio frame.
- Number of bits by frame may be configurable.

Specific Feature: Companding Mode (1/2)

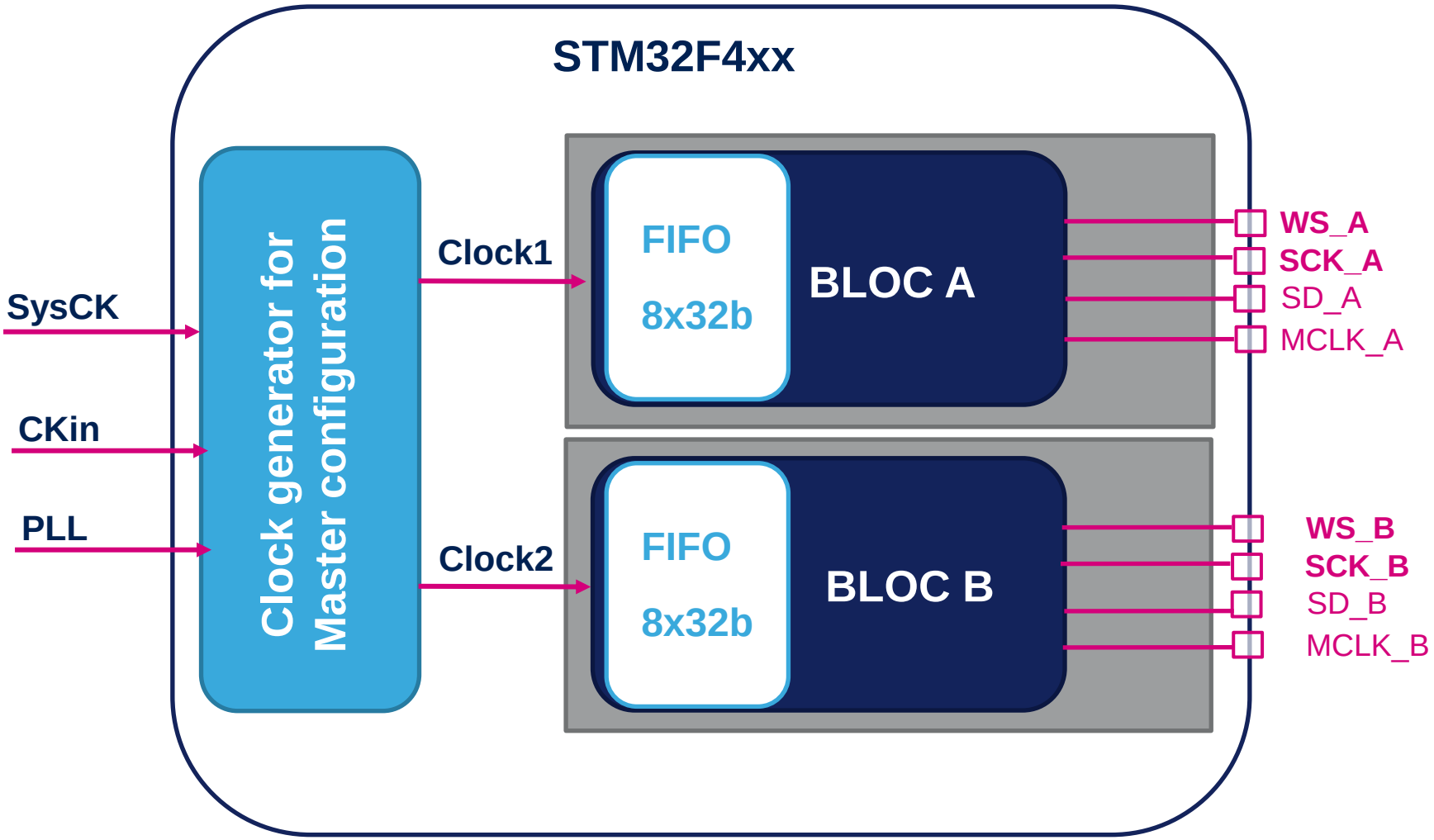
- Two companding modes are supported: **µ-Law** and the **A-Law** log which are a part of the CCITT G.711 recommendation
- The companding standard employed in the United States and Japan is the **µ-Law** and **allows 14 bits** bits of dynamic range
- The European companding standard is **A-Law and allows 13 bits** of dynamic range.
- The µ-Law and A-Law formats encode data into 8-bit code elements with **MSB alignment**. ➔Companded data is **always 8 bits wide**
- Companding standard (µ-Law or A-Law) can be computed based on 1's complement or 2's complement representation depending on the CPL bit setting in the SAI_xCR2 register.

  ➔ **Not applicable when AC'97 selected.**

SAI Features(3/3)

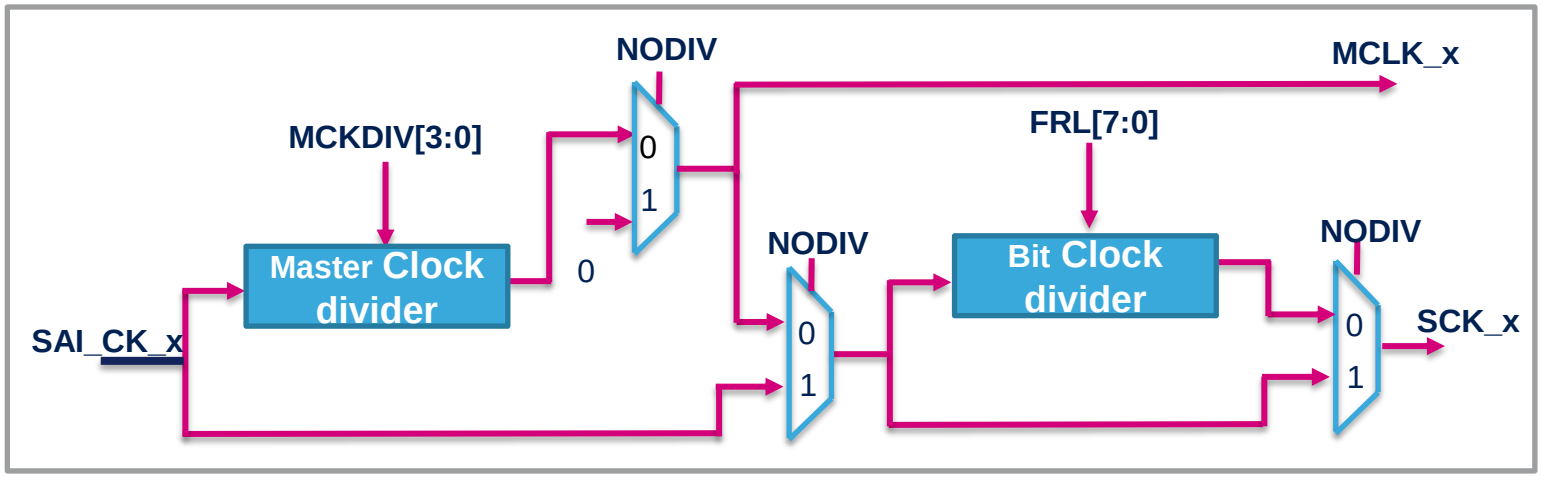
- Frame synchronization active level configurable:
  - Offset,
  - Bit length,
  - Level
- DMA interface with 2 dedicated channels to handle access to the dedicated integrated FIFO of each SAI audio sub-block.

SAI Top Level Architecture



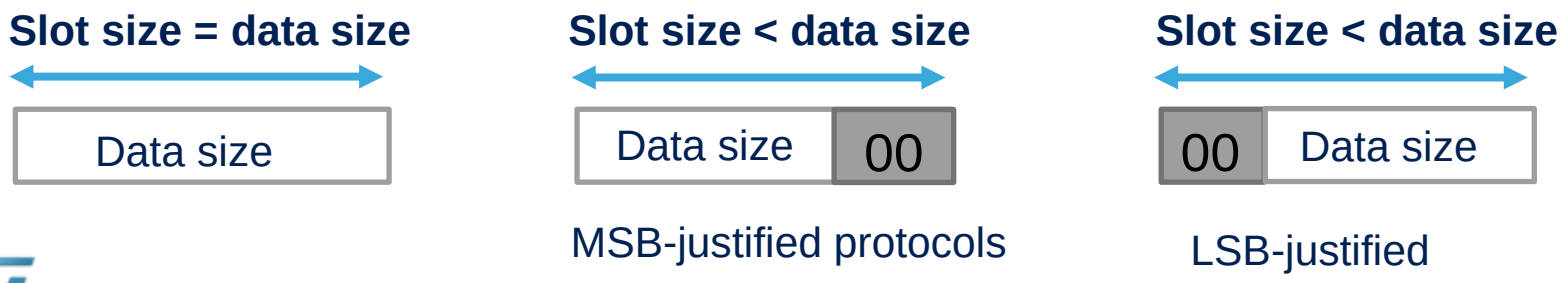
Audio Clock configuration

- Each audio block has its **own clock generator.**
- The clock source for the clock generator(SAI_CK_x) is derived from: I2S PLL, SAI PLL or External Clock(I2S_CKIN pin).
- MCLK/FS ratio is **fixed to 256**



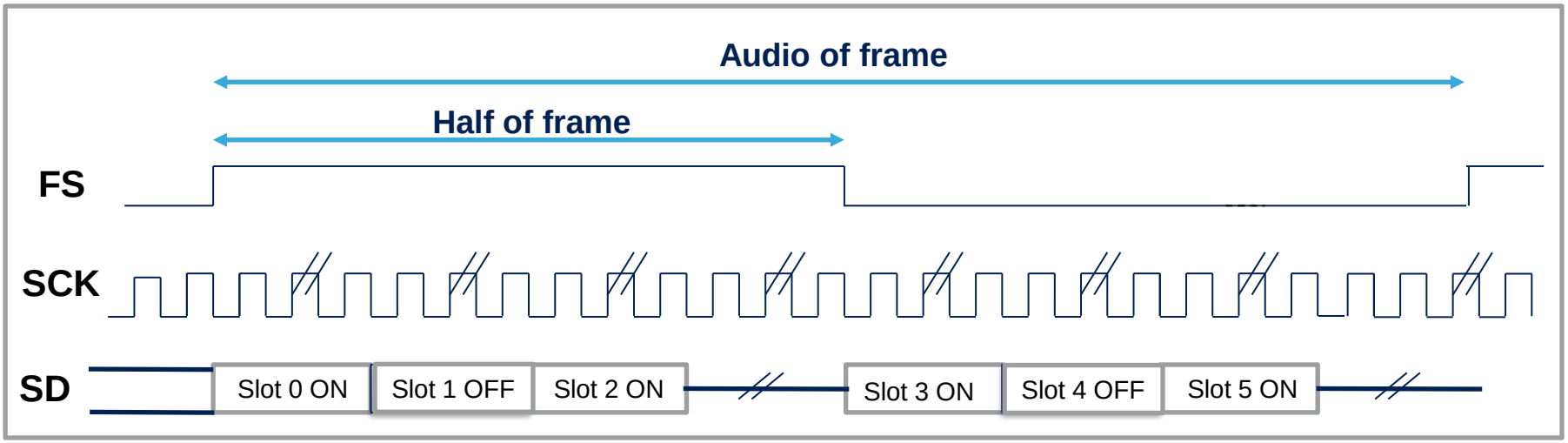
Audio Slot configuration(1/2)

- The maximum number of slots per audio frame **is fixed to 16**
  - Configured through bits “NBSLOT[3:0] in the SAI_xSLOTR register +1”.
- For **AC’97 protocol** the number of slots **is automatically set to 13 slot**
  - *The value of NBSLOT[3:0] is ignored.*
- Each slot can be defined as a **valid slot, or not**
  - By setting bit SLOTEN[15:0] in the SAI_xSLOTR register.
- The **size of the slots** is selected by setting bit SLOTSZ[1:0] in the SAI_xSLOTR register.



Audio Slot configuration(2/2)

- During the transfer of a non-valid slot, **0 value** will be forced on the data line **or** the SD data line will be released to **HI-z** depends of TRIS bit setting in the SAI_xCR2 register
  - ➔No request to read or write the FIFO linked to this inactive slot



STM32F29 Smart Digital Peripherals

life.augmented

STM32F429 series

DSP & FPU Hands-On Lab

life.augmented

Hands-On

Floating Point Hardware versus Floating Point Emulation

life.augmented

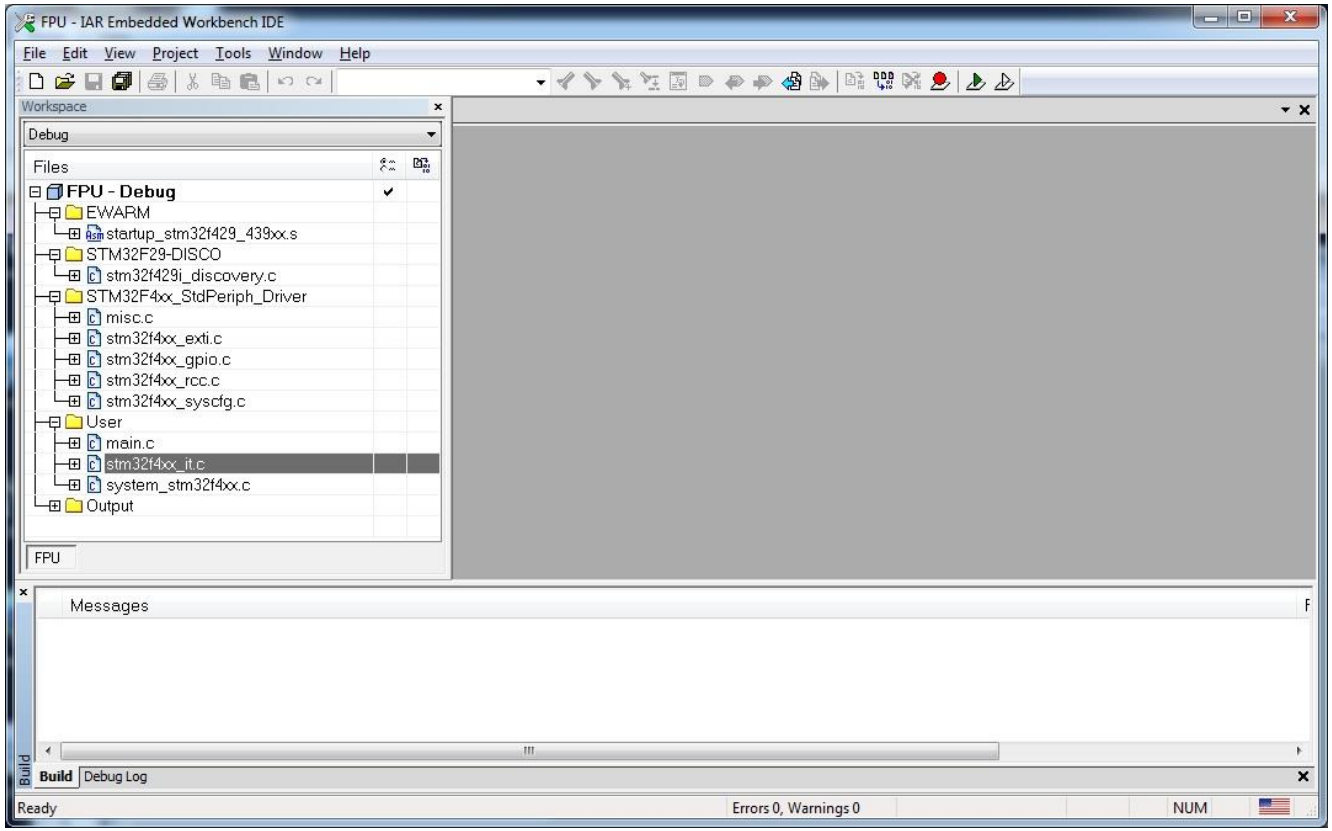
BenchMarking Floating Point Calculations

- The floating point equations below are in the FPU Example of the STM32F429I-Discovery_FW_V1.0.0 library.
- Hands-on Training II will Bench Mark calculations using the FPU.
  - Code Size:
  - Execution:
    - Counting instructions using the EWARM simulator.
    - Using the STM32 SysTick Counter to count CPU cycles.
    - Measuring the toggle time of an I/O line on the STM32F429Discovery_Kit.
    - How could we count cycles direct on the STM32F429Discovery Kit? Hint SWO.

```
/* Initialze float variables */
Data0 = 1.0 ;
Data1 = 2.0 ;

/* Start a set of mathematical operations */
Result = fabsf(Data0) ;

Result = -Data0 ;

Result = Data1 + Data2 ;

Result = Data1 - Data2 ;

Result = Data0 + Data1 * Data2 ;
Result = Data0 - Data1 * Data2 ;
Result = -Data0 + Data1 * Data2 ;
Result = -Data0 - Data1 * Data2 ;

Result = fmaf(Data1,Data2,Data0) ;
Result = fmaf(-Data1,Data2,Data0) ;
Result = fmaf(Data1,Data2,-Data0) ;
Result = fmaf(-Data1,Data2,-Data0) ;

Result = Data1 / Data2 ;
Result = sqrtf(Data1) ;
Result = sqrt(Data1) ;
Result = fabs(Data1) ;
```

Step #1 Activate the FPU Project

- Close any other active project.
- Select the FPU Project located at this location:



STM32F429I-Discovery_FW_V1.0.0\Projects\Peripheral_Examples\FPU_Example\EWARM\FPU.EWW

Step #2a FPU code size BenchMark

- Generate a MAP output file.
  - Project → Options→ Linker → List → ‘Generate Linker map file.
- Enable the FPU hardware.
  - Project → Options→ General Options → FPU = VFPv4.
- Set IAR Optimizer to ‘NONE’.
  - Project → Options→ C/C++ Compiler → Optimizations = None.
- Enable the ST-Link Debugger.
  - Project → Options → Debugger →Setup → Driver = ST-LINK.
- Edit main.c: Go to line 53, to view the equations being solved.
- Build the project.
  - Project → ReBuild All.

Step #2b FPU code size BenchMark

With HW FPU

- Open the FPU.Map file.
  - Record the project's FLASH and SRAM size

```
  5 028 bytes of readonly  code memory
     68 bytes of readonly  data memory
  1 052 bytes of readwrite data memory
```

- Disable the Hardware FPU.
  - Project → Options→ General Options → FPU = none.
- Build the project
  - Project → ReBuild All.

Without HW FPU

- Open the FPU.Map file.
  - Record the project's FLASH and SRAM size
- Discuss the results.

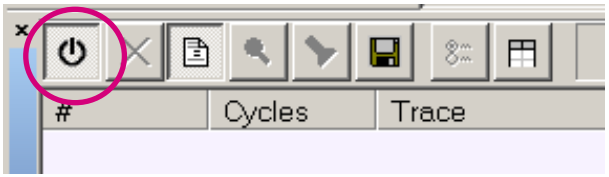
```
  5 644 bytes of readonly  code memory
     68 bytes of readonly  data memory
  1 052 bytes of readwrite data memory
```

Step #3a FPU performance Bench Mark, Simulator

- Close the FPU.MAP file
- Enable FPU hardware.
  - Project → Options→ General Options → FPU = VFPv4.
- Verify the IAR Optimizer is set to ‘NONE’.
  - Project → Options→ C/C++ Compiler → Optimizations = None.
- Enable the EWARM Simulator
  - Project → Options → Debugger →Setup → Simulator.
- Build the project.
  - Project → ReBuild All.
- Start the IAR EWARM Simulator
  - Project → Download and Debug (<ctrl> F5).

Step #3b FPU performance Bench Mark, Simulator

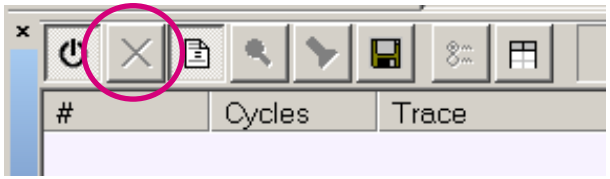
- Place a BREAK point at Line 72.
- Open a ‘Variable’ Window to view the results. Over this window on the Disassembly window.
  - View → Auto.
- Close the ‘Messages’ Window.
- Open the EWARM Simulator Trace Window.
  - Simulator → Trace.
- Activate the Simulator Trace.



- The FPU Project is setup in a while(1) which will loop for ever. Each time through, the execution will hit the BREAK point set on Line 72 and STOP so the results can be viewed.

Step #3c FPU performance Bench Mark, Simulator

- Run the Program.
  - Debug → GO or F5.
- The execution will stop at the BREAK point. Clear the Simulator Trace Window contents.

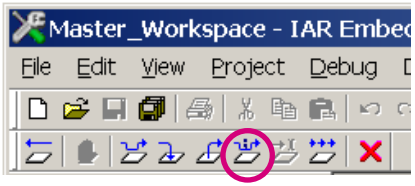


- Run the Program.
  - Debug → GO or F5
- The Simulator Trace window will show the END cycle count.
  - Record the end cycle count value.
  - Browse to the start of the Simulator Trace window data and look for the instruction, Data0 = 1.0. Record the cycle count value.
  - Subtract the two cycle count values to get the FPU execution cycle count.

22352494 - 22350914➔ 1578 instructions to complete

Step #3d FPU performance Bench Mark, Simulator

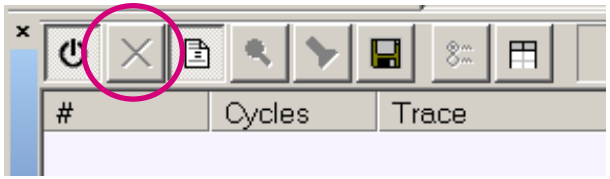
- Single Step through the code and take note of the Auto window to view the results of the executed statement.
  - Debug → Next Statement OR



- Stop the Debugger/Simulator.
  - Debug → Stop Debugging Or <ctrl><shift><del>F5
- Disable the FPU.
  - Project → Options→ General Options → FPU = none.
- The Simulator is still enabled, so just Download and Debug.
  - Project → Download and Debug or <ctrl>D
    - The code will recompile and link and download. The simulator is already enabled.
- The code did not change, the BREAK point is still at Line 72.

Step #3e FPU performance Bench Mark, Simulator

- Run the Program.
  - Debug → GO or F5.
- The execution will stop at the BREAK point. Clear the Simulator Trace Window contents.



- Run the Program.
  - Debug → GO or F5
- The Simulator Trace window will show the END cycle count.
  - Record the end cycle count value.
  - Browse to the start of the Simulator Trace window data and look for the instruction, Data0 = 1.0. Record the cycle count value.
  - Subtract the two cycle count values to get the FPU execution cycle count.

446129 – 443590 ➔ 2539 instructions to complete

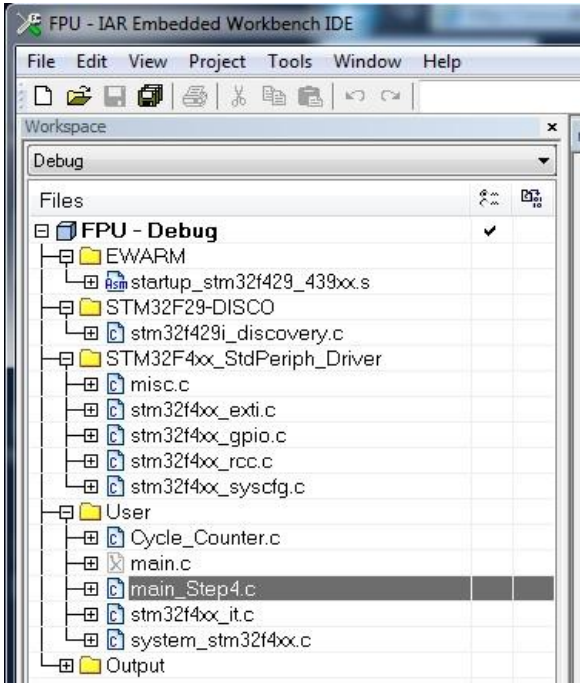
Step #4a FPU performance Bench Mark, SysTick

- STOP the Debugger.
- Enable FPU hardware.
  - Project → Options→ General Options → FPU = VFPv4.
- Verify the IAR Optimizer is set to ‘NONE’.
  - Project → Options→ C/C++ Compiler → Optimizations = None.
- Enable the STLINK Debugger
  - Project → Options → Debugger →Setup → STLINK
  - Project → Options → Debugger →Download… ‘Verify Download, Use FLASH Loader’.
  - Project → Options → ST-LINK →Interface… ‘SWD’.
- Modify the FPU Project source by copying main_step4.c, Cycle_Counter.c, and Cycle_Counter.h into the FPU Project ‘working directory’. The instructor will show you the ‘working directory’.
  - These files are located in →

C:\STM32Seminar\STM32F429I-Discovery_FW_V1.0.0\Projects\FPU_BenchMark_Counter

Step #4b FPU performance Bench Mark, SysTick

- Add main_step4.c and Cycle_Counter.c to the User folder of the FPU Project.
  - Highlight User
  - right mouse click → Add→ Add files….



- ‘Exclude from build’ the original main.c file
  - Highlight main.c
  - right mouse click → Options → ‘Exclude from build’

- Build the project.
  - Project → Rebuild All.

- Download and Debug to the STM32F429Discovery board.
  - Project → Download and Debug (<ctrl> F5).

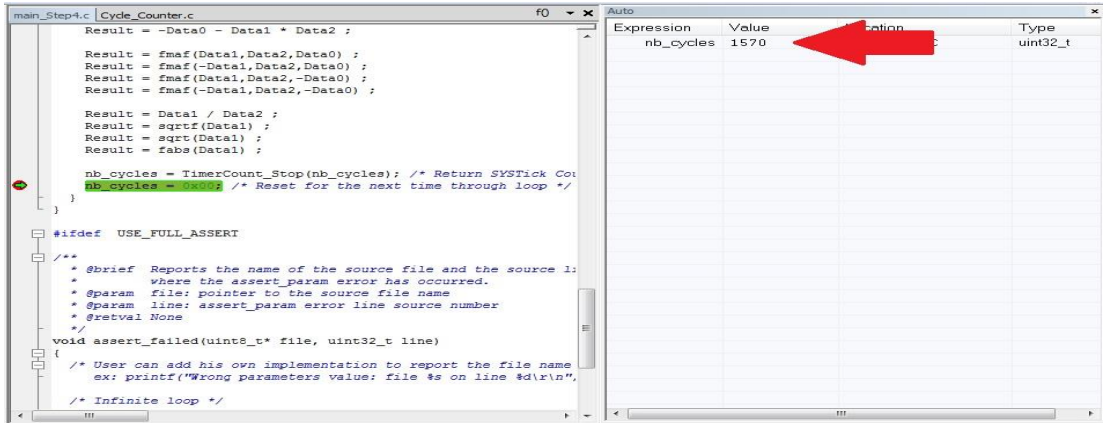
- The Debugger will open main_step4.c and break at line 56.

Step #4c FPU performance Bench Mark, SysTick

- Explore the following functions in main_Step4.c file:
  - Line 79: TimerCount_Start().
  - Line 110: TimerCount_Stop(nb_cycles)
    - What is happening with the SysTick?
    - What Frequency is the SysTick operating at?
  - Discuss the location of each function in the program and what will be counted.
- Place a BREAK point at Line 111.
- Open a ‘Variable’ Window to view the results. Overlay the ‘Auto’ window on the Disassembly window.
  - View → Auto.
- The FPU Project is setup in a while(1) which will loop for ever. Each time through, the execution will hit the BREAK point set on Line 111 and STOP so the results can be viewed.

Step #4d FPU performance Bench Mark, SysTick

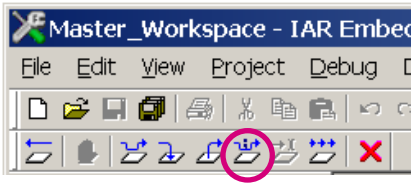
- Execute one loop of the FPU Project.
  - Debug → GO or F5.
  - The execution will stop at the BREAK point, line 111.
- What is the value of ‘nb_cycles’?
- Calculate and record the execution time.



- 1570 cycles @ 180Mhz ➔ 8.7 usec.

Step #4e FPU performance Bench Mark, SysTick

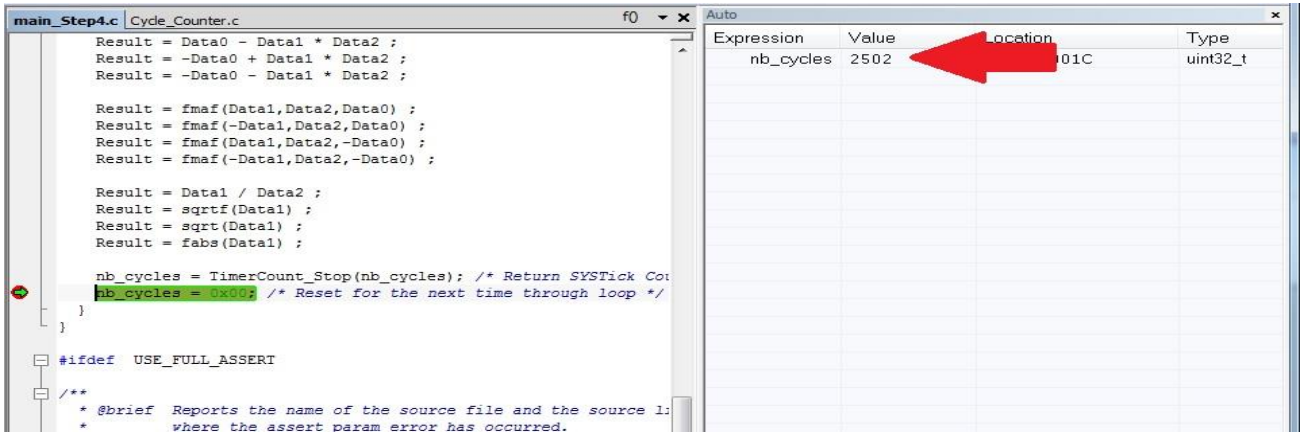
- Single Step through the code and take note of the Auto window to view the results of the executed statement.
  - Debug → Next Statement OR



- Stop the Debugger
  - Debug → Stop Debugging Or <ctrl><shift><del>F5
- Disable the FPU.
  - Project → Options→ General Options → FPU = none.
- The STLINK is still enabled, so Download and Debug.
  - Project → Download and Debug or <ctrl>D
  - The code will automatically Rebuild before downloading.
- The code did not change, the Debug opens main_step4.c at line 67, and the BREAK point is still at Line 111.

Step #4f FPU performance Bench Mark, SysTick

- Execute one loop of the FPU Project.
  - Debug → GO or F5.
  - The execution will stop at the BREAK point, line 111.
- What is the value of ‘nb_cycles’?
- Calculate and record the execution time.



- 2502 cycles @ 180Mhz ➔ 13.9 usec.

Step #5 FPU performance Bench Mark, I/O Toggle

- Stop the Debugger/Simulator.
  - Debug → Stop Debugging Or <ctrl><shift><del>F5
- Enable the ST-LINK Debugger and Disable the EWARM Simulator.
  - Project → Options → Debugger →Setup → ST-LINK
- Open main.c, go to line 72: STM_EVAL_LEDToggle(LED4);
  - Using EWARM, what physical I/O line is this function toggling.
  - Connect an O-Scope probe to PG14.
- Using the previous sequence of steps (3 thru 3d), measure the toggle rate of LED4 for the FPU Enabled and the FPU Disabled (emulation).

FPU Bench Marking Summary

- **FPU Code Density Results**
  - FPU Enabled → Code FLASH = 5028, Data Flash = 68, SRAM = 1052
  - FPU Disabled → Code FLASH = 5644, Data Flash = 68, SRAM = 1052
  - FPU Improvement ➔ 11%

FPU Performance Results	IAR Simulator Counts	SysTick Counts	I/O Toggle
FPU Enabled	1578	1570 @180Mhz 8.7 usec	8.4 usec
FPU Disabled	2539	2502 @180Mhz 13.9 usec	14.2 usec
FPU Improvement	38%	37%	40%

Thank you

www.st.com/stm32f429idiscovery

STM32F429-Discovery
Gyro Hands On

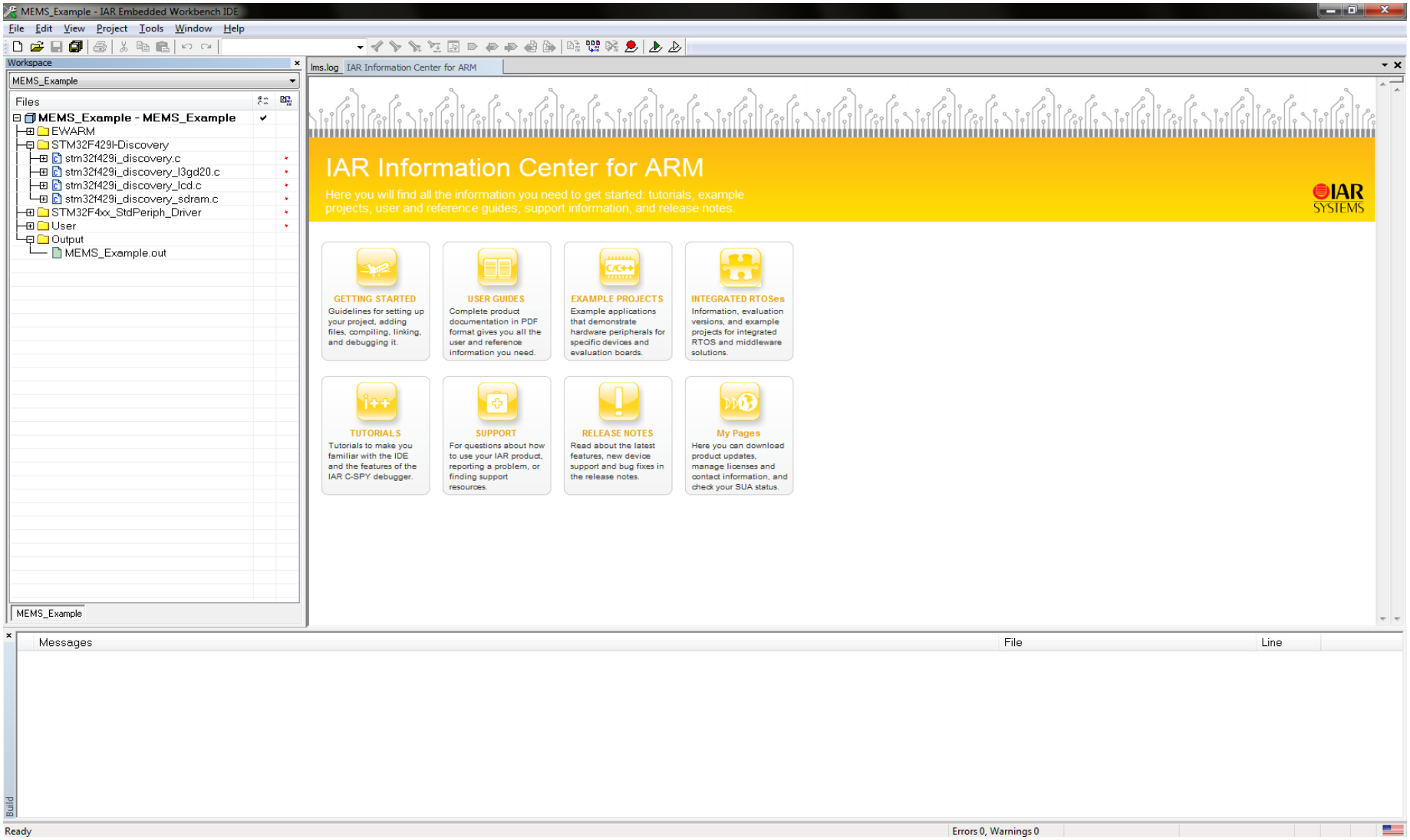
life.augmented

STM32F429 Discovery Kit Gyro

- This board is equipped with the ST Microelectronics 3-Axis electronic gyro, L3GD20.

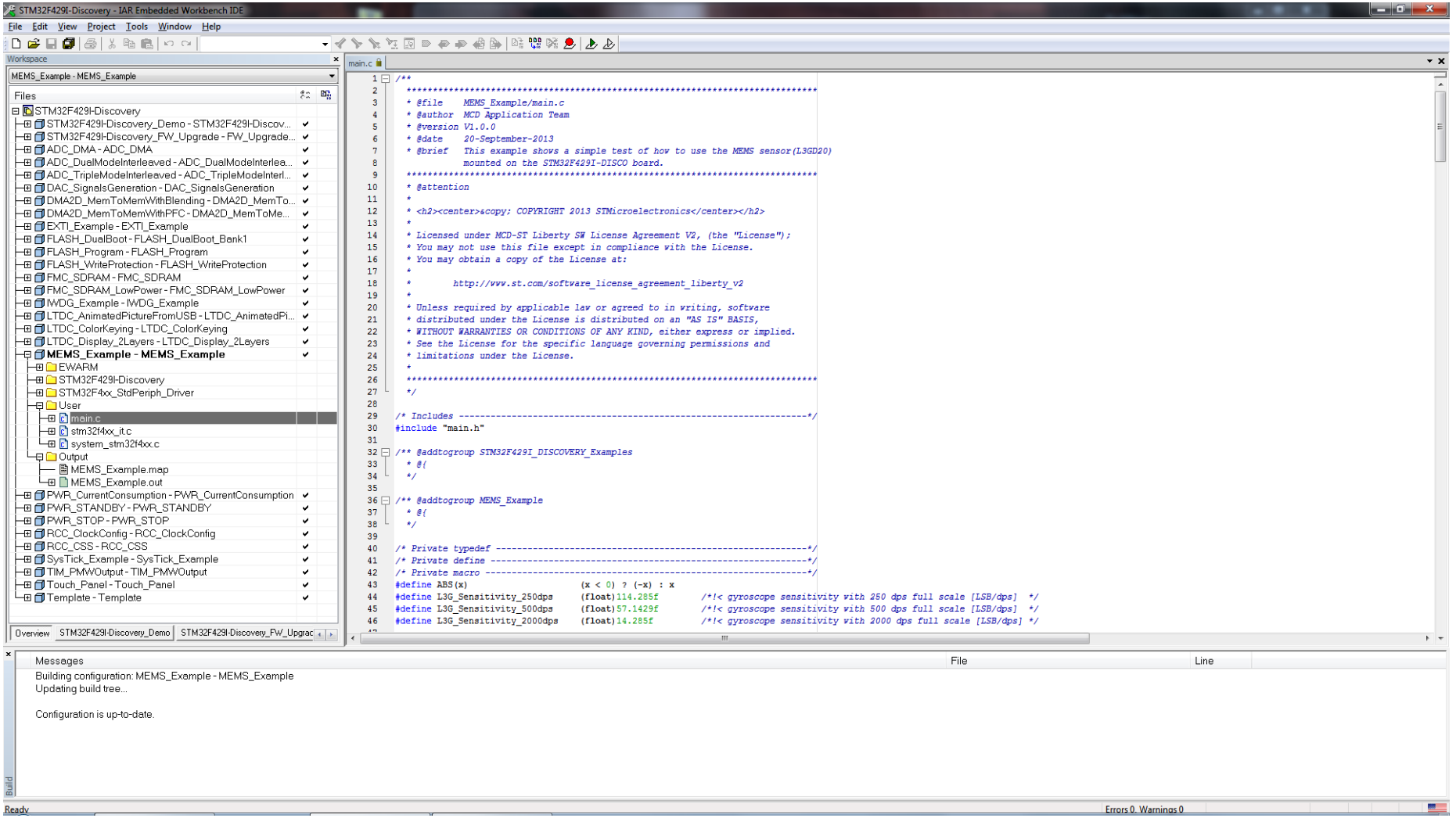
- It includes a sensing element and a digital I2C/SPI interface providing the measured angular rate with a selectable full scale of +/-250, +/-500, or +/-2000 degrees per second

STM32F429 Discovery Kit Gyro



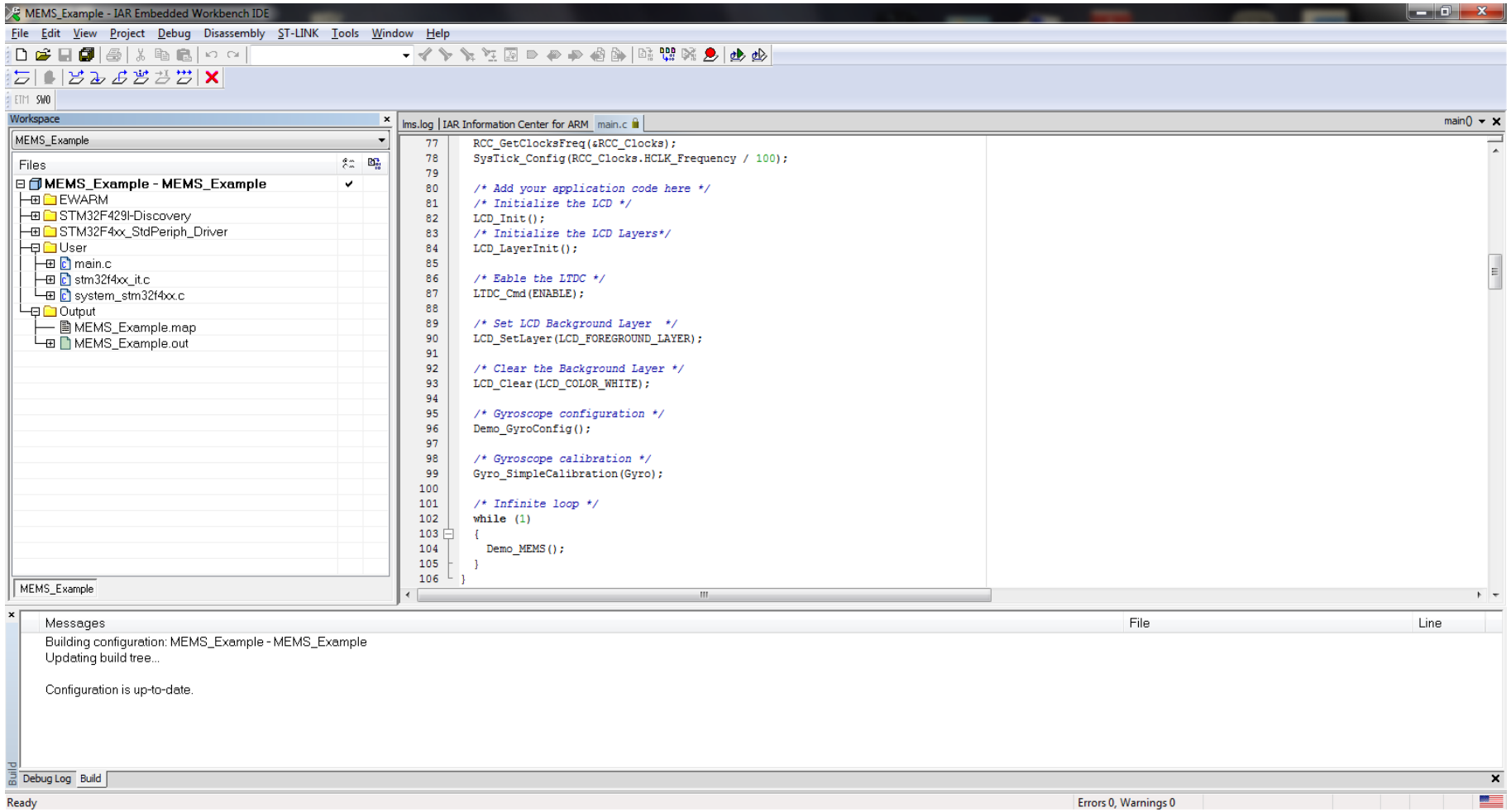
life.augmented

STM32F429 Discovery Kit Gyro



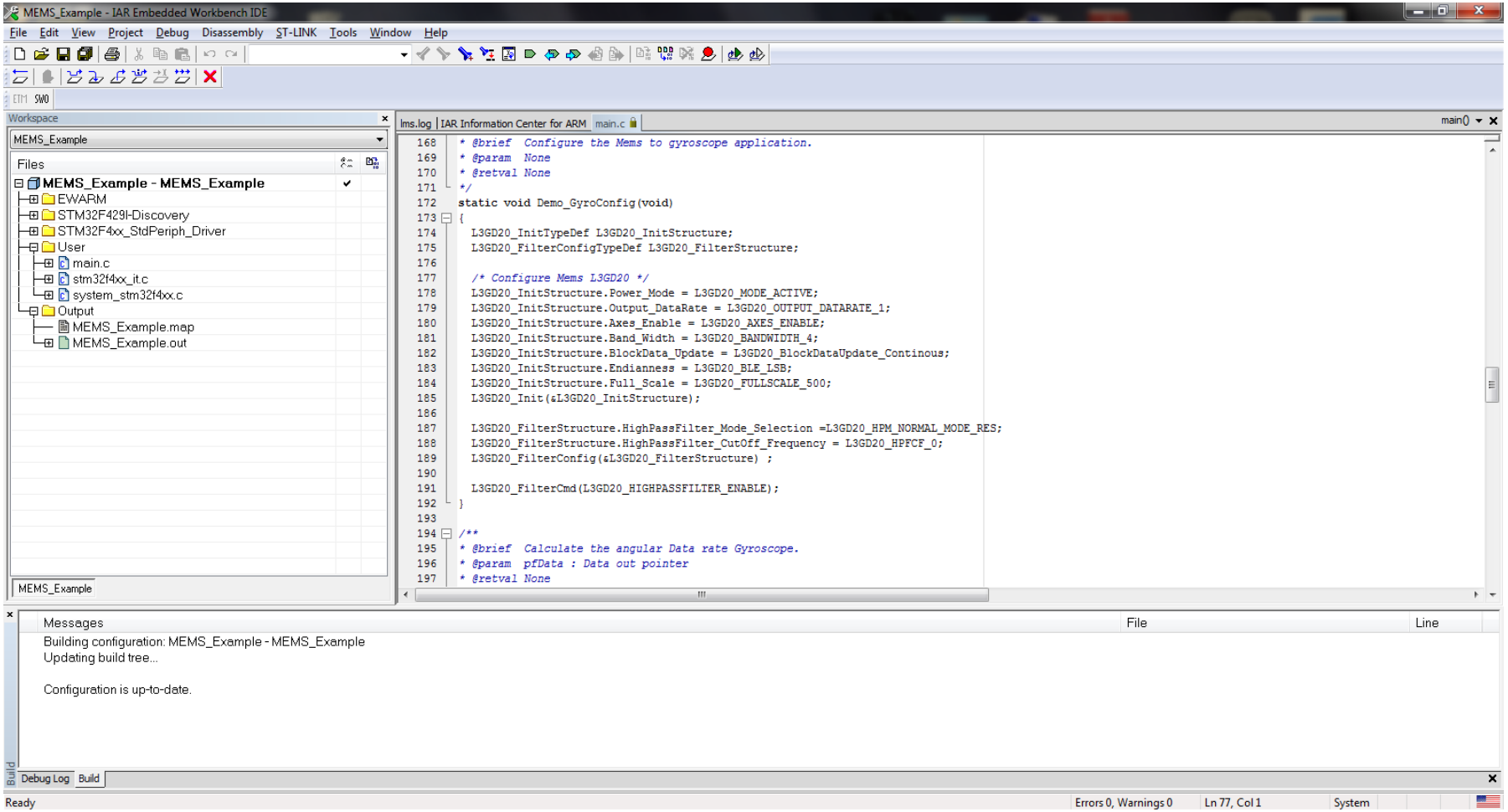
life.augmented

STM32F429 Discovery Kit Gyro



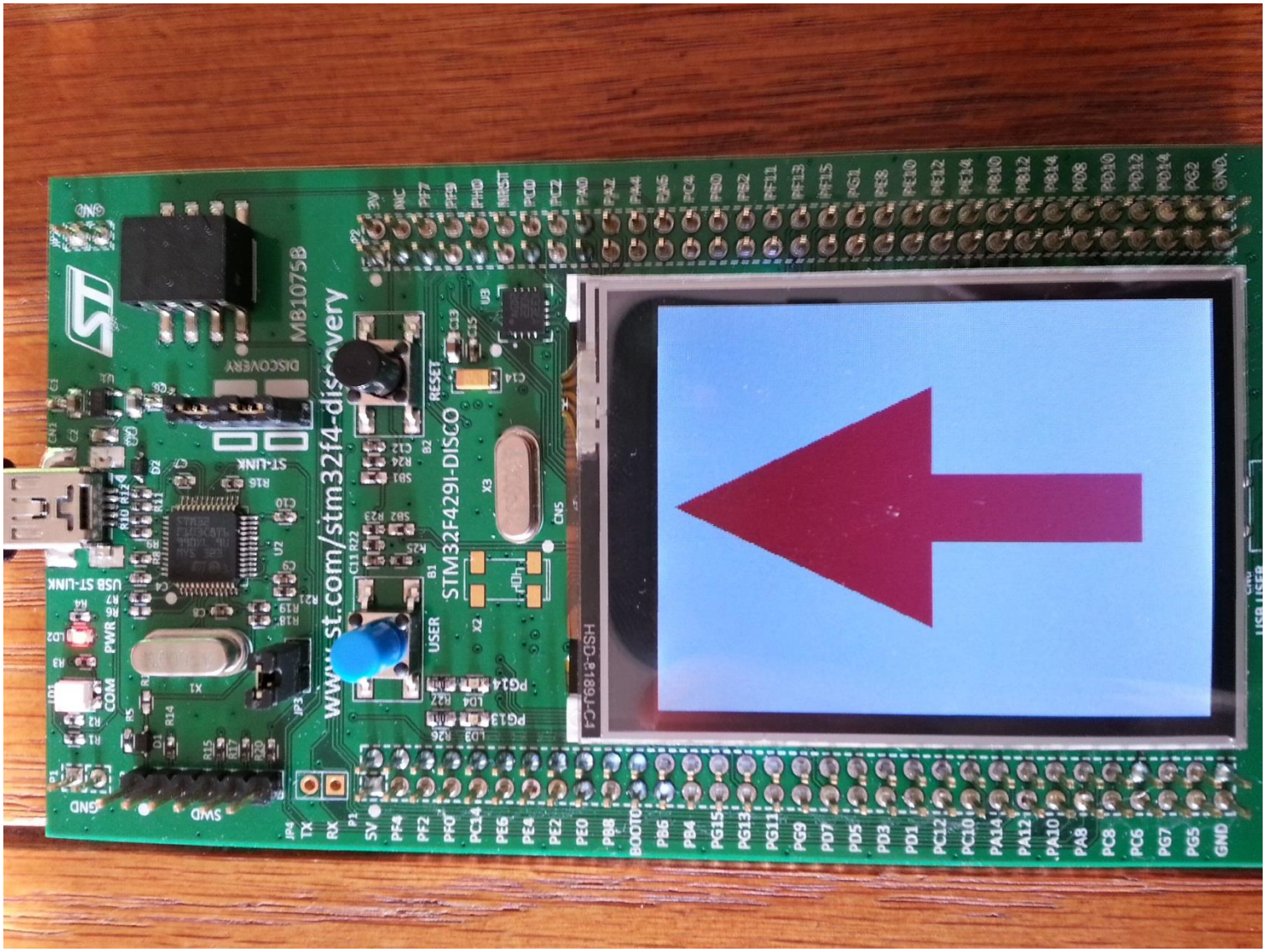
life.augmented

STM32F429 Discovery Kit Gyro



life.augmented

STM32F429 Discovery Kit Gyro



STM32 Ecosystem and Software Development Suites

life.augmented

A large community of partners

Hardware Development Tools (1/2)

- A wide offering of Debugging Probes and Programming Tools
  - ST ST-LINK/V2, low cost in circuit debugger/programmer
  - Raisonance Rlink
  - IAR I-jet and JTAGjet-Trace
  - ARM/Keil ULink
  - …

- Ability to evaluate completely our STM32 series:
  - via our Evaluation Boards
  - via partners boards, for example IAR, ARM/Keil or Raisonance

Hardware Development Tools (2/2)

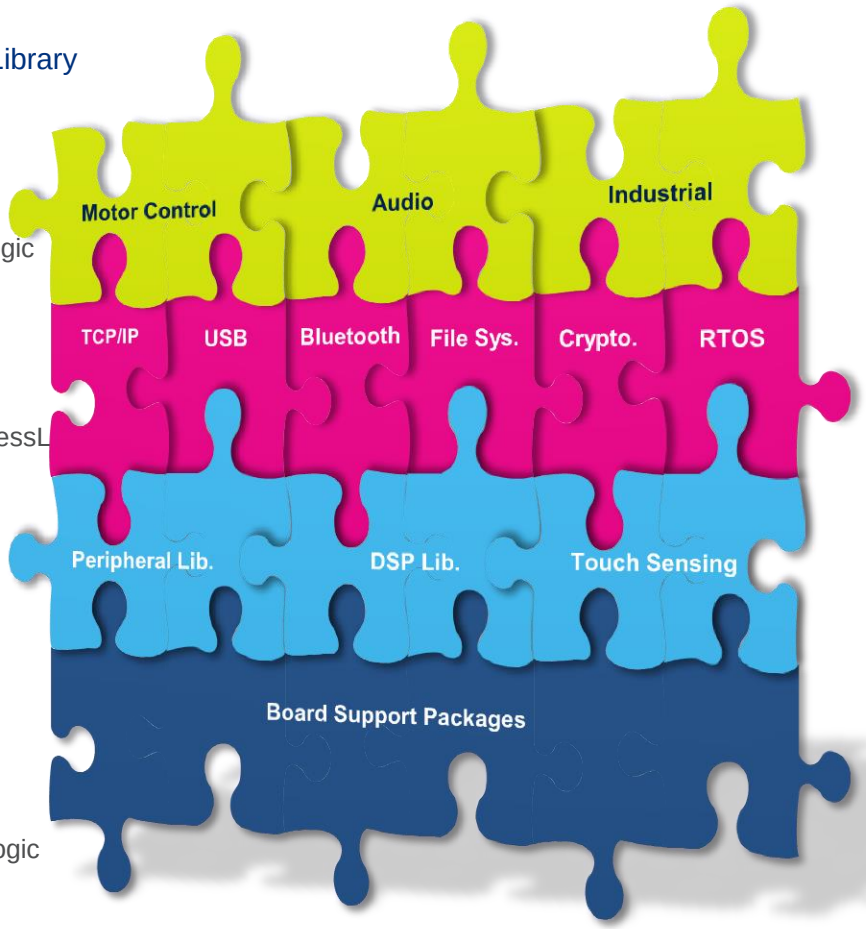
- Low-cost evaluation boards
  - ST Discovery Kits
  - Raisonance Primers
  - Hitex Kits

  …

- Open-Hardware Initiatives
  - Arduino-compatible, for example Leaflabs Maple, Olimexino-STM32, SecretLabs Netduino
  - Gadget for example Mountaineer, CHI Fez-Cerberus,

Embedded Software (Firmware) (1/3)

- Full STM32 coverage in standard C language, from low level drivers to Firmware stacks and Application bricks
- Low level:
  - Free ST Boards Support Packages (BSP)
  - Free ST Peripheral Libraries (Drivers) and Free ST DSP Library
- Firmware Stacks:
  - RTOS
    - Open source for example FreeRTOS
    - Commercial solutions, for example Micrium or ExpressLogic
  - Cryptographic
    - Free ST “STM32 Cryptographic Library”
  - USB
    - Free ST USB Libraries
    - Commercial solutions for example HCC, Micrium or ExpressL
  - TCP/IP
    - Open source for example uIP or LwIP
    - Commercial solutions for example Interniche or Quadros
  - File Systems
    - Open source for example FatFS
    - Commercial solutions for example HCC or SEGGER
  - BlueTooth
    - Commercial solutions for example Alpwise or Searan
  - Graphics
    - Free ST “STemWin” graphical library
    - Commercial solutions for example SEGGER or ExpressLogic
  - Touch sensing
    - Free ST STMTouch Library



Embedded Software (Firmware) (2/3)

- Still in C language, ST also proposes some high level application bricks, in chosen application fields:
  - Motor Control: Free full solution for Field Oriented Control (FOC) drive of 3-phase Permanent Magnet Synchronous Motors (PMSM)
    - ST PMSM FOC Library (Firmware)
    - ST Motor Control Workbench: Graphical Interface, for a complete and easy customization of Library.
  - Industrial: extensive partner network, covering widely many Industrial protocols like EtherCat, ProfiNet, Modbus, DeviceNet, …
  - Audio: Extensive offer, **optimized** to take benefit from Cortex-M3 or Cortex-M4 cores:
    - ST Audio Codecs: MP3, WMA, AAC-LC, HE-AACv1, HE-AACv2, Ogg Vorbis, ….
    - ST Vocal Codecs: G711, G726, IMA-ADPCM, Speex, …
    - ST Post Processing Algorithms:
      - Sample Rate Converters, any ratio or optimized “standard ratios” like 48KHz to 24KHz
      - Filters with examples like Bass Mix, Loudness….
      - Smart Volume Control ensuring a volume increase with no saturation
      - Stereo Widening

      …

Embedded Software (Firmware) (3/3)

- Beyond legacy development in C language, ST and its partners is proposing also virtual machines-based languages or model-based approaches, like:

Java Language

STM32Java solution, featuring:
- Specific part numbers
- A collection of embedded software
- A full development tool under Eclipse

.Net Micro Frameworks

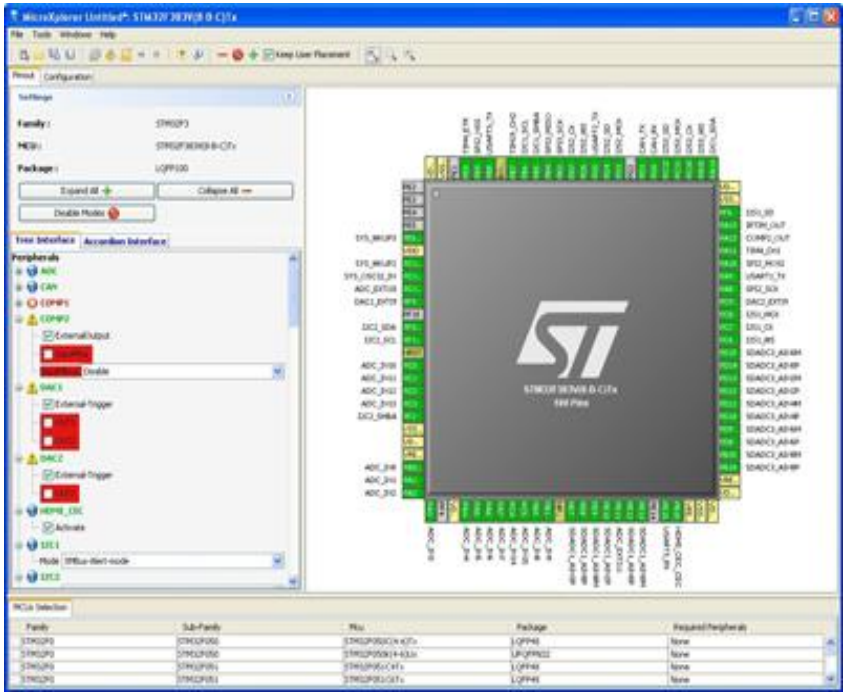
- Through open source solutions supporting fully of STM32:
- With the very standard Microsoft Visual Studio as development environment

Matlab/Simulink

- Code generation for models taking benefit from ARM Cortex-M DSP instructions, delivered ST MathWorks
- Modeling of STM32 peripherals, delivered by ST

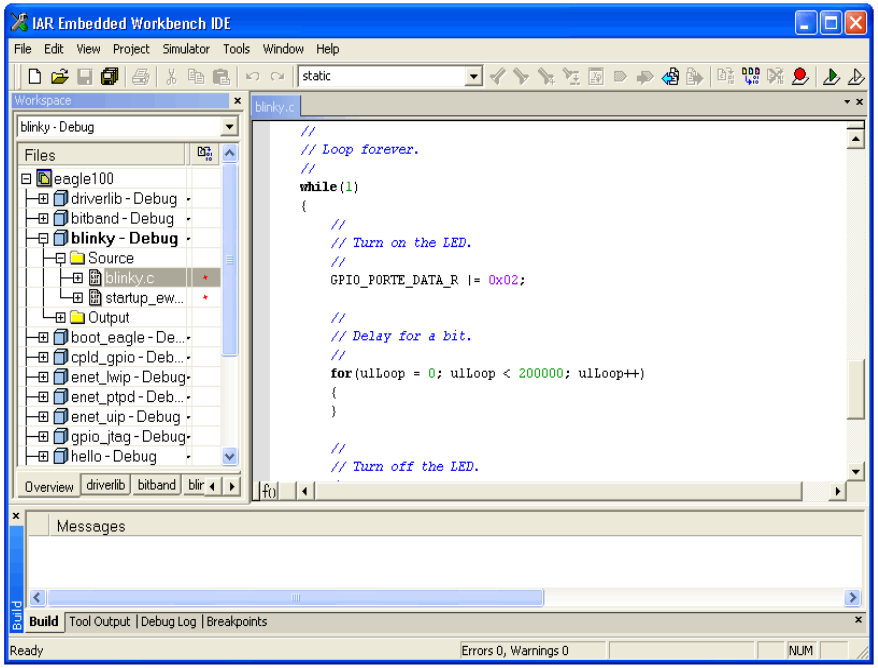
Software Development Tools

- Free ST MicroXplorer Configuration Tool
  - Makes it easy to select the most appropriate STM32 from the broad portfolio
  - Allows the user to check how the package pins will be configured, detecting potential conflicts, using a graphical interface
  - Generates code for IOs initializations
  - Power Consumption wizard
  - Regularly extended with new features



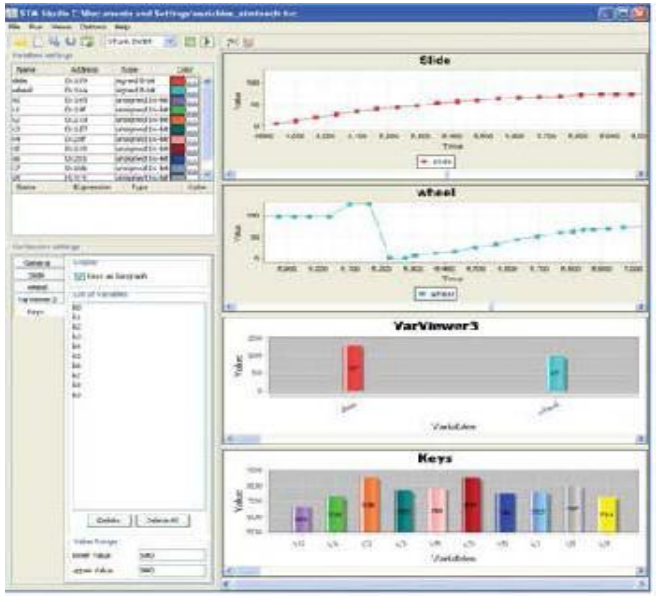
Software Development Tools (2/3)

- Broad range of Development Toolchains directly supporting the STM32 family
- Many open source solutions exist, while Commercial solutions all have some free options, either time-limited or code size-limited
- Commercial solutions:
  - IAR EWARM
  - Keil MDK
  - Atollic TrueStudio
  - Rowley CrossWorks
  - Raisonance Ride
  - Altium Tasking
  - Emprog Thunderbench
  - …
- Free or Open source solutions:
  - Embest CooCox
  - SEGGER emIDE
  - Yagarto
  - …



Software Development Tools (3/3)

- Free ST Monitoring Tool STMStudio
  - Supports low cost STLink-V2 debugging probe
  - Ability to select any global variable to be monitored by just providing the compiled file (elf)
  - Several acquisition methods:
    - 100% non-intrusive option !
    - Application-synchronized option
  - Ability to monitor the behavior of chosen variables, through a collection of graphical widgets



STM32 tools

Starter and promotion kits
Numerous boards

Motor control kit

Range of evaluation boards

STM32 W evaluation kit STM32W-SK

STM32 promotion kits

EvoPrimer

STM32-ComStick

5 Discovery kits

More than **15** different development IDE solutions

IAR SYSTEMS, KEIL Tools by ARM, atollic, i SYSTEM, RAISONANCE, TASKING, Green Hills SOFTWARE, SIGNUM SYSTEMS, RA, SEGGER, hitex DEVELOPMENT TOOLS, LAUTERBACH DEVELOPMENT TOOLS, Altium, aiji SYSTEM 아이지시스템, Embest

More than **25** different RTOS and stack solution providers

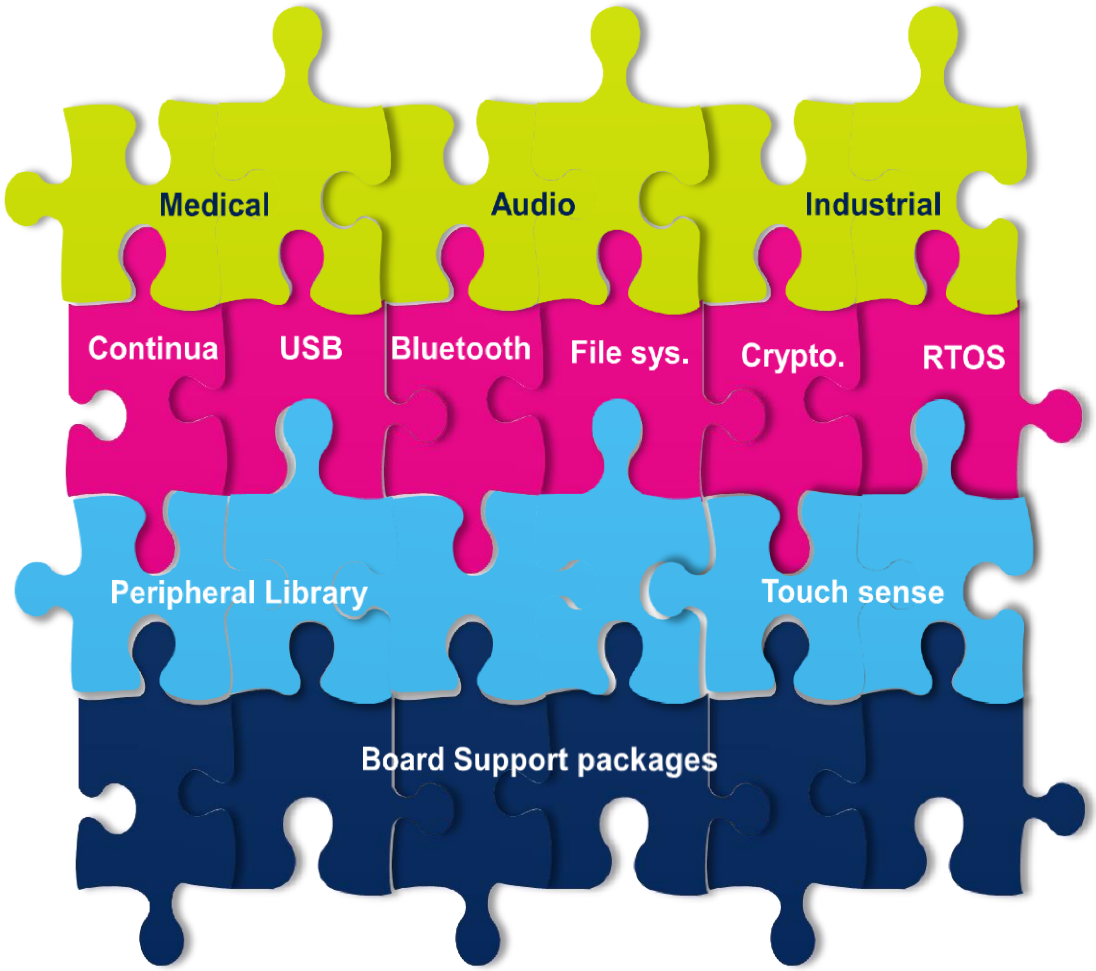
HighIntegritySystems, interniche technologies, inc., KEIL Tools by ARM, SEGGER, Quadros Systems Inc., eCosCentric, CMX SYSTEMS, VDE, Green Hills SOFTWARE, freeRTOS, port, IXXAT, MESCO Engineering, embeX, ANDREA, expresslogic, ALPWISE, µd Micro Digital, embedded solutions, Micrium, MicroControl Systemhaus für Automatisierung, embeddedlabs, RoweBots, JUNGO Connectivity Software, ARC CORE The future is open, AVIX-RT, HCC embedded

life.augmented

STM32 Firmware solutions

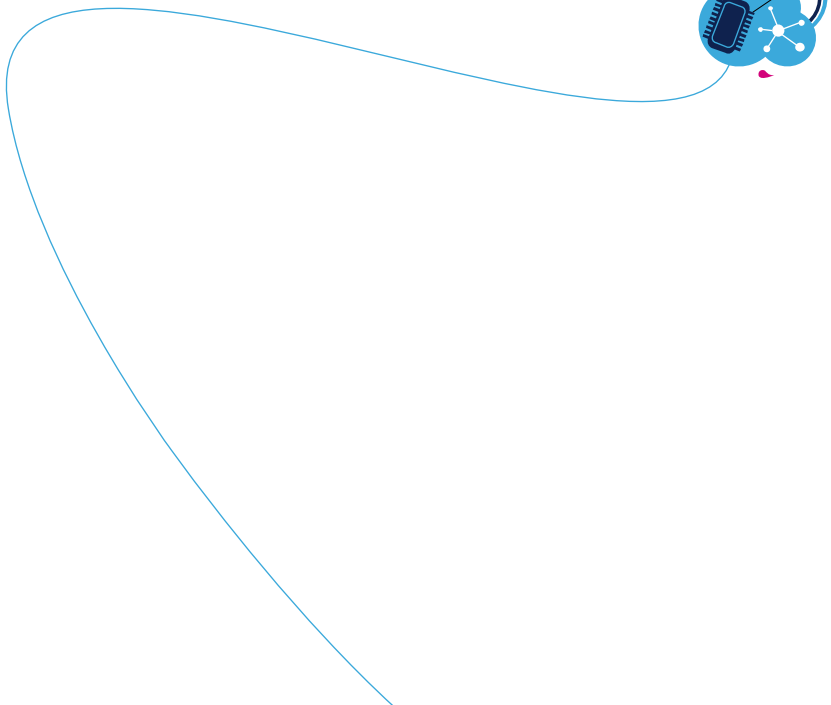
More than just silicon

- **Free solutions**
  - From ST
  - From open source
- **Large 3rd party network**
  - More than 30 companies with STM32 solutions



life.augmented

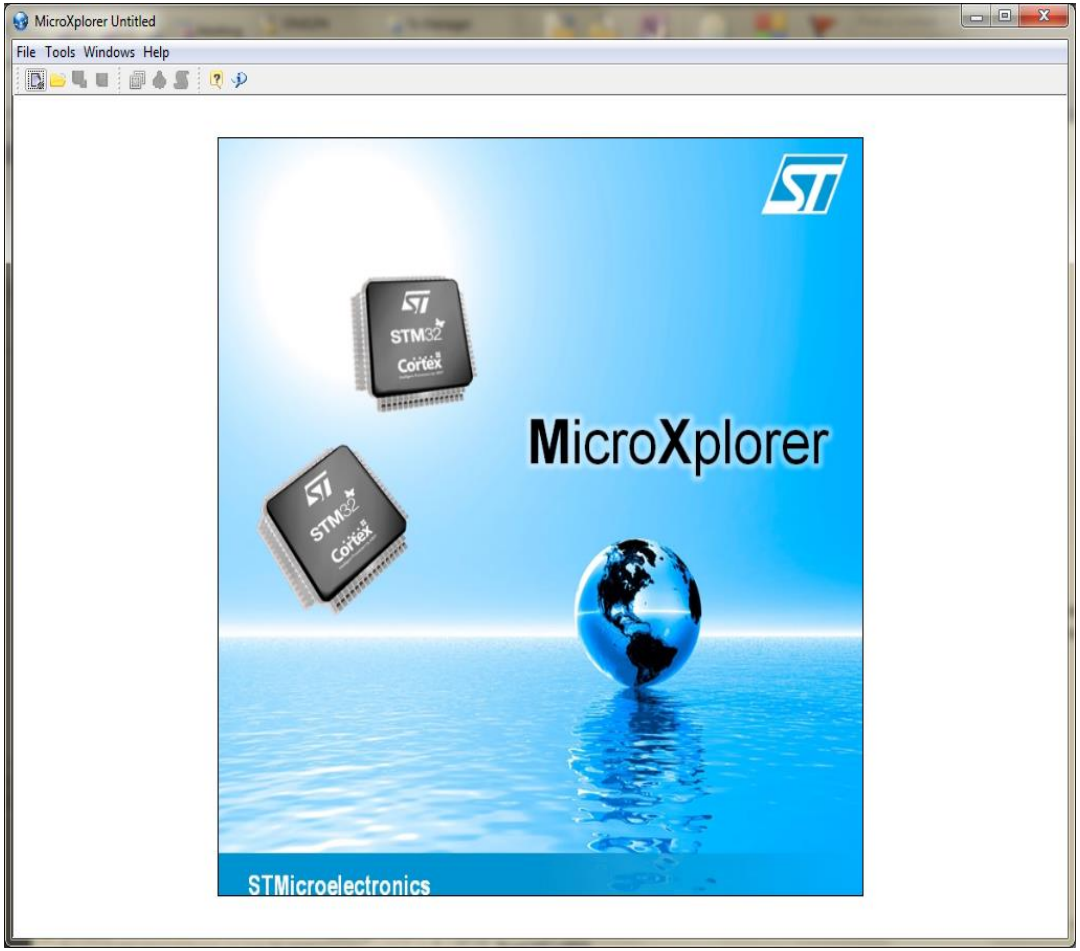
STM32F429 Discovery MicroXplorer



life.augmented

ST MicroXplorer v3.2

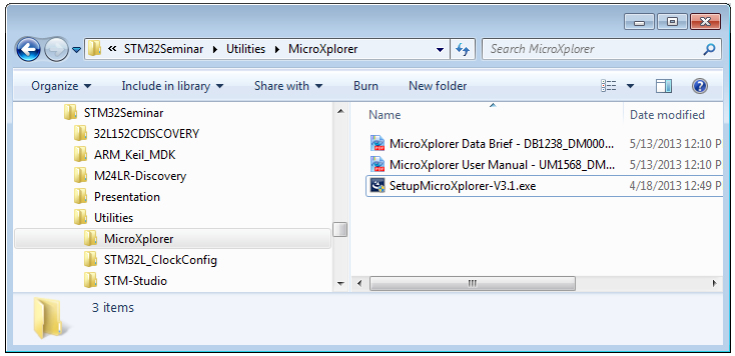
- MicroXplorer 3.2 is an intuitive graphical tool for the STM32 MCU families

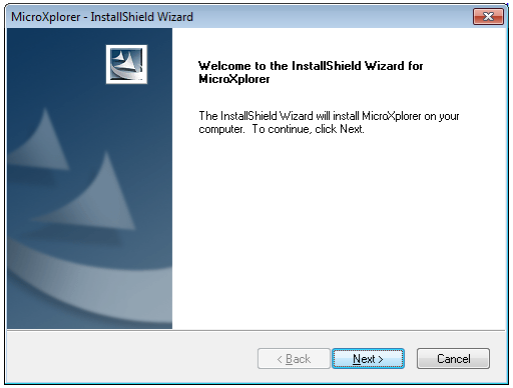


  - Easy pin out configuration via the selection of the peripherals required by the application
  - GPIO configuration (input, output or peripheral alternate function) and generation of corresponding initialization code
  - Power consumption calculation for STM32L low-power family

Tool Installation

- From the USB Image that you copied to your PC, find the following folder and launch the MicroXplorer installer
- **C:\STM32Seminar\Utilities\MicroXplorer\InstallShieldMicroXplorer-V3.2.exe**



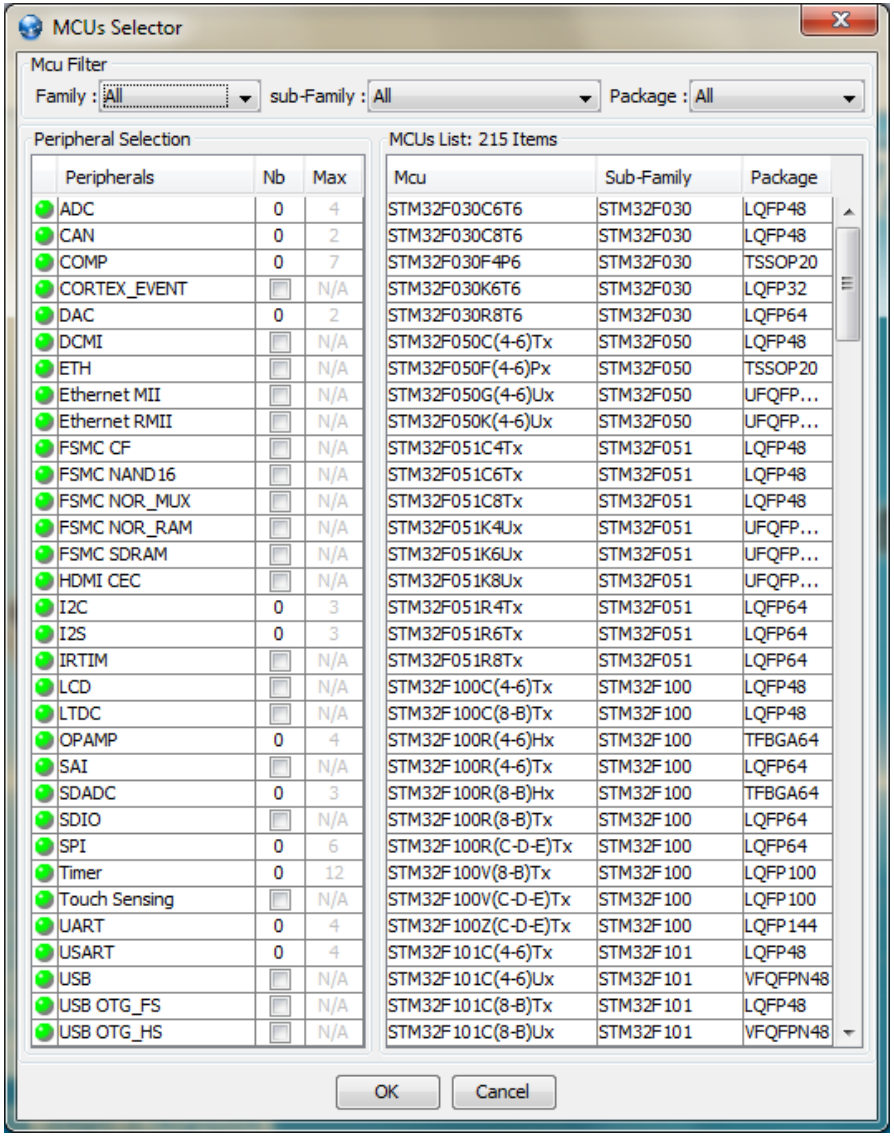


- Note: This software needs **Java Runtime v6.7 or higher**. If you have an older Java version, use one of the two Java installers at this path:
  - **C:\STM32Seminar\Utilities\Java**

ST MicroXplorer v3.2

TOOLS -> MCU's Selector

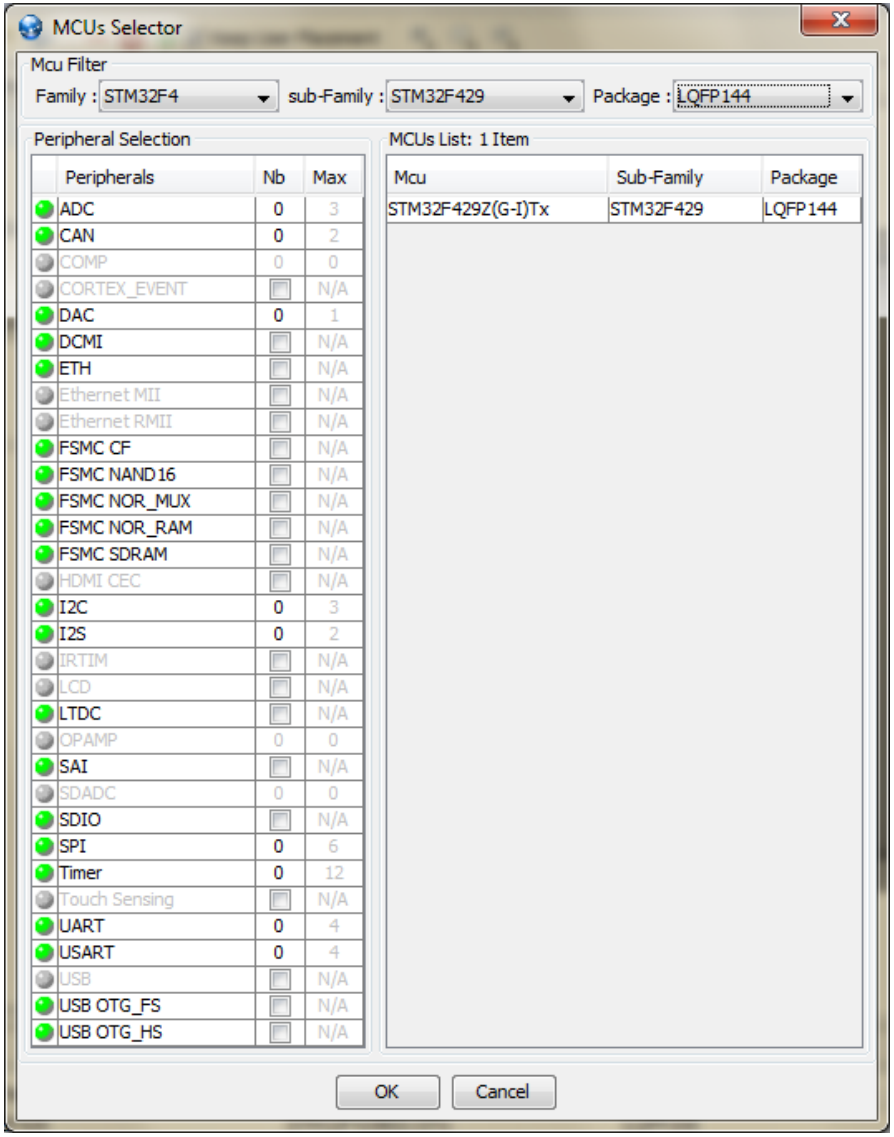
- Family: F0, F1, F2, F3, F4, L1, or All
- Sub-Family: many choices, i.e. STM32F429
- Package: choices for this device are LQFP100, LQFP144, LQFP176, LQFP208, TFBGA216, UFBGA176



ST MicroXplorer v3.2

TOOLS -> MCU's Selector

- Family: F0, F1, F2, F3, F4, L1, or All
- Sub-Family: STM32F429
- Package: choices for this device are LQFP100, LQFP144, LQFP176, LQFP208, TFBGA216, UFBGA176

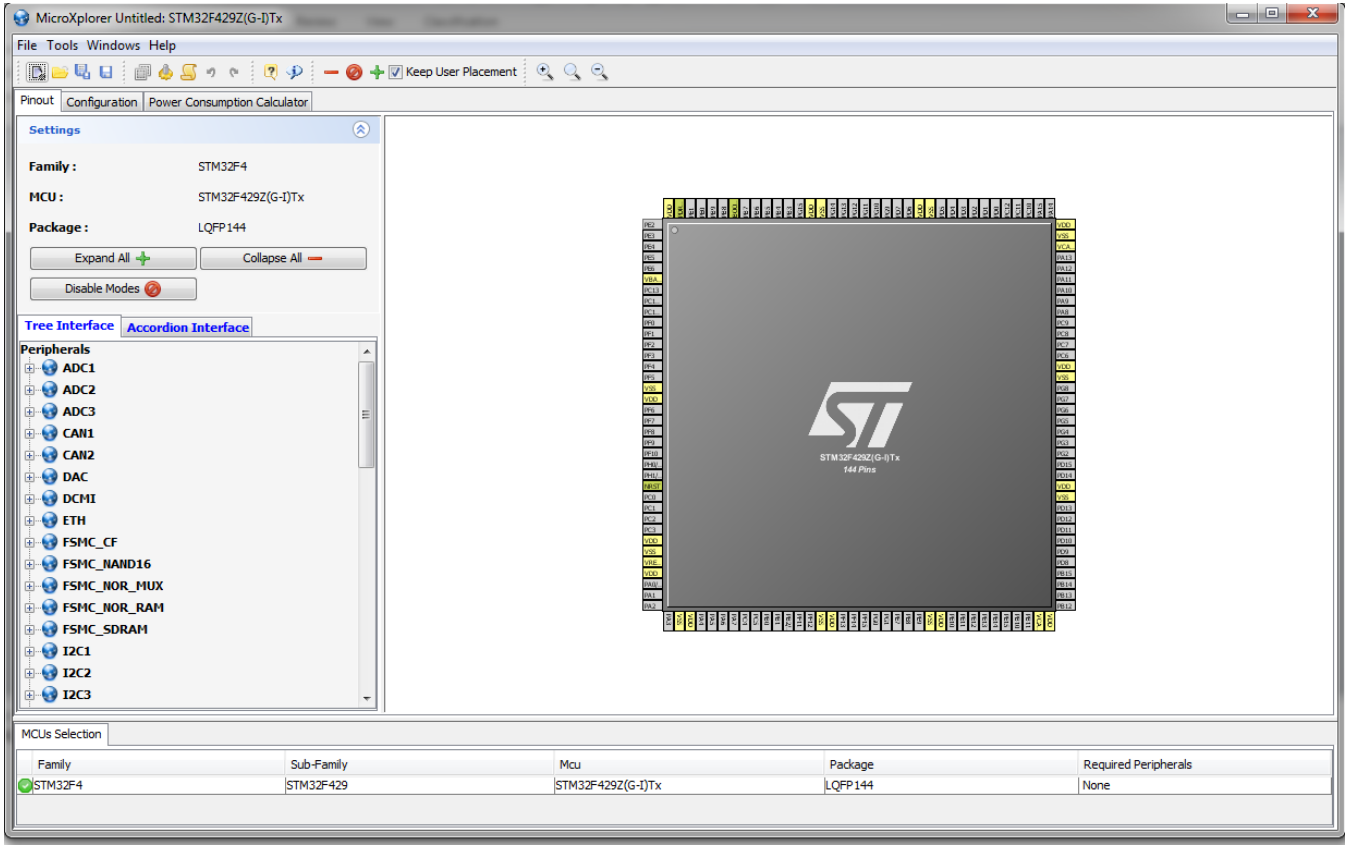


life.augmented

ST MicroXplorer v3.2

Pin assignment: Peripherals needed for our application (STM32F4 Discovery Kit):

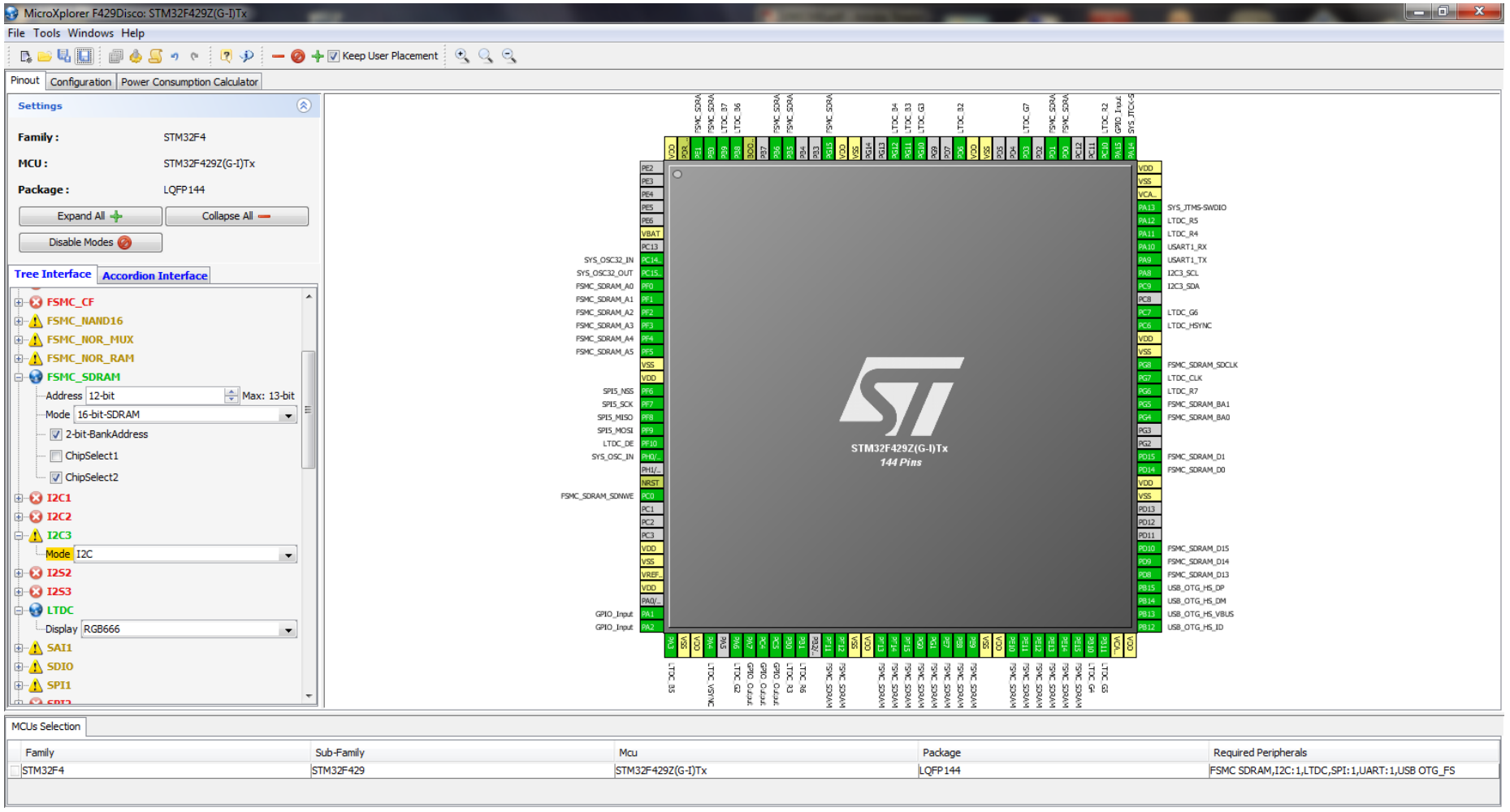
- FSMC SDRAM
- 1-I2C
- LTDC
- 1-SPI
- 1-UART
- USB OTG_FS



ST MicroXplorer v3.2

- Pin assignment: Peripherals needed for our application with their full configuration (STM32F429 Discovery Kit):
  - FSMC_SDRAM: 12 bit address, 16 bit data, check 2 bit bank address, check ChipSelect2
  - I2C3: I2C
  - LTDC: RGB666
  - SPI5: Full Duplex Master with NSS
  - SYS: Debug = Serial Wire, OSC = HSE External Oscillator, OSC32 = LSE External Oscillator
  - USART1: Asynchronous

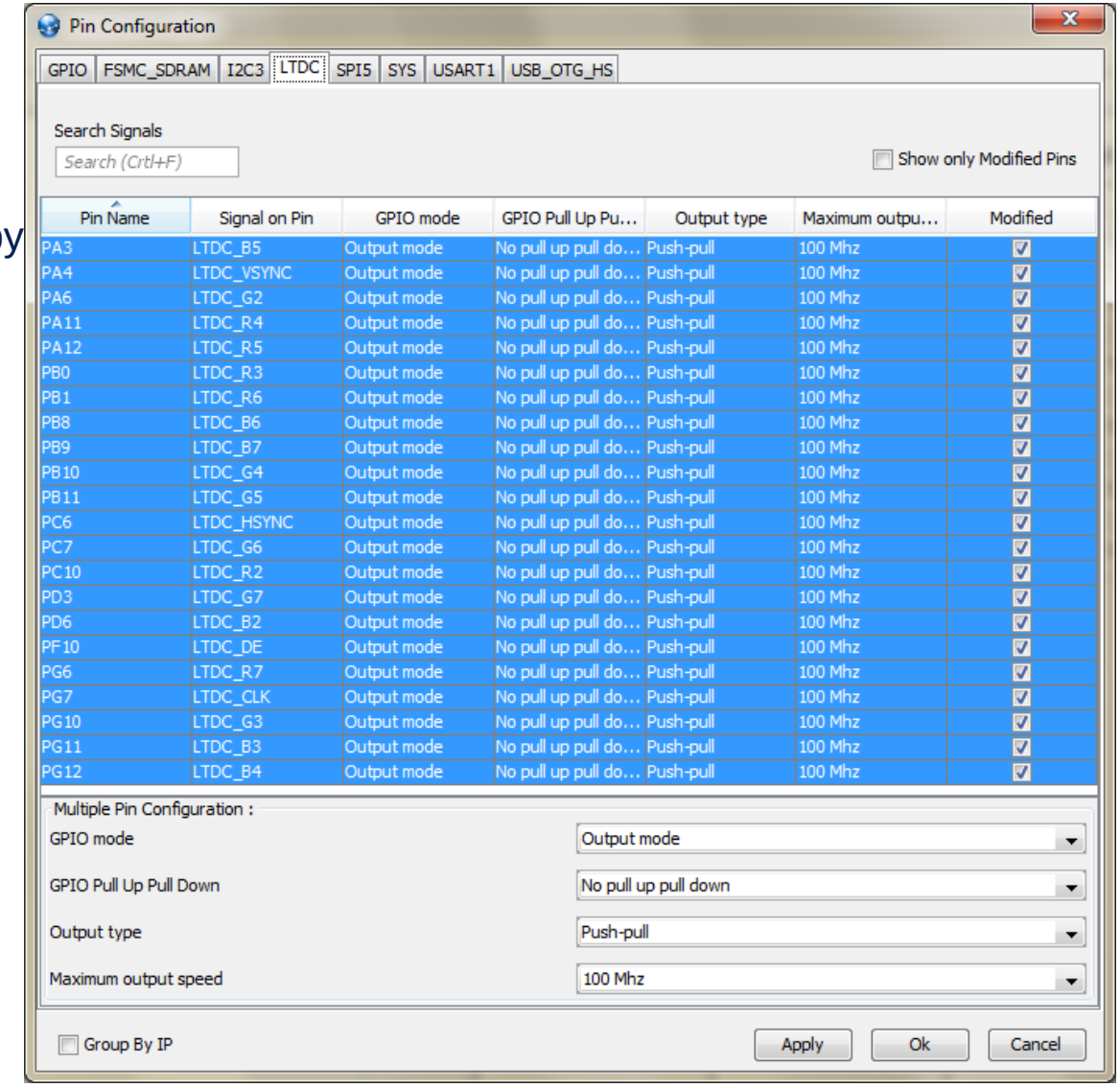
ST MicroXplorer v3.2



life.augmented

ST MicroXplorer v3.2

- CONFIGURATION Tab
- Tabs show peripheral
- Configuration by each pin or by
- GPIO Modes
  - Input
  - Output
  - Alternate Function
  - Analog input
- Pull up/down/none
- Output type
  - Push-Pull
  - Open Drain
- Maximum Output Speed
  - 2-25-50-100MHz



life.augmented

ST MicroXplorer v3.2

TOOLS -> Generate Report

- .pdf for later documentation
- .txt if .pdf is not adequate

TOOLS -> Generate Code

- src
  - mx_gpio.c
  - mx_main.c
- inc
  - mx_gpio.h

```
/** I2C3 GPIO Configuration
          PC9------> I2C3_SDA
          PA8------> I2C3_SCL   */
/*Enable or disable the AHB1 peripheral clock*/
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC|RCC_AHB1Peri
ph_GPIOA, ENABLE);
/*Configure GPIO pin */
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_9;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_Init(GPIOC, &GPIO_InitStruct);
/*Configure GPIO pin */
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_8;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_Init(GPIOA, &GPIO_InitStruct);
/*Configure GPIO pin alternate function */
GPIO_PinAFConfig(GPIOC, GPIO_PinSource9, GPIO_AF_I2C3);
/*Configure GPIO pin alternate function */
GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_I2C3);
```