• Introduction to STMicro

• Overview of STM32 Nucleo Ecosystem

• Introduction to STM32Cube solution

• STM32Cube FW package presentation
  • Exploring the package content
  • Versioning and maintenance model
  • Documentation update for the STM32Cube F4

• Hardware Abstraction Layer
  • HAL overview
  • System peripherals HAL drivers overview (RCC, GPIO, DMA, Cortex, PWR)
  • Standard peripheral HAL driver model
  • Guidelines for writing a HAL example

• Demo of STM32CubeMX PC software too

• Presentation and Demo of mbed
A global semiconductor leader
The largest European semiconductor company
2014 revenues of $7.40B
Approximately 43,600 employees worldwide
Approximately 8,700 people working in R&D
11 manufacturing sites
Listed on New York Stock Exchange, Euronext Paris and Borsa Italiana, Milano

As of December 31, 2014
Our MEMS & Sensors are augmenting the consumer experience.

Our automotive products are making driving safer, greener and more entertaining.

Our digital consumer products are powering the augmented digital lifestyle.

Our Microcontrollers are everywhere making everything smarter and more secure.

Our smart power products are allowing our mobile products to operate longer and making more of our energy resources.
Effective Q4 2014, DCG and IBP product groups merged to form DPG.
Flexible and Independent Manufacturing
An unwavering Commitment to R&D

Advanced research and development centers around the globe

~ 15,000 patents; ~9,000 patent families; more than 500 new filings (in 2014)

~ 8,700 people working in R&D and product design

As of December 31, 2014
Partners with our Customers worldwide

79 sales offices in 35 countries
ST’s vision and strategy

OUR VISION

Everywhere microelectronics make a positive contribution to people’s lives, ST is there.

OUR STRATEGY

Leadership in Sense & Power, Automotive Products and Embedded Processing Solutions

OUR 5 GROWTH DRIVERS

- Smart Power
- MEMS and Sensors
- Automotive
- Digital Consumer & ASICs
- Microcontrollers
STM32 Nucleo Ecosystem

Electronics made easy!

Edoardo Gallizio
edoardo.gallizio@st.com
It’s a great Opportunity

Connected Devices (“Nodes”)

Mobile devices

Hubs and Gateways

Augmented Things

Sense
Process
Connect
Translate
Power

Billion Units Installed Base

2013
13 Billion

2020
36 Billion
The Building Blocks are already here

Sensors & Actuators
- Motion MEMS
- Environmental Sensors
- MEMS microphones
- Touch Sensor
- Micro-actuators
- Proximity sensor
- Image sensors

Processing
- Low-power brain
- Sensor fusion

Communication
- Ultra-low power connectivity

Interfaces
- Analog

Power
- Power and energy management
IoT Products & ST offering

ST has a unique portfolio with all the key technologies and products

**Sensors**
- Ultra-low power connectivity

**Ultra Low Power Microcontrollers**
- Analog and mixed signal components

**Smart Energy Management**
- Dev. Tools

**Smart City**
- Smart Street Lights
- Smart Meters
- Environment smart node
- Smart parking systems

**Smart Home**
- Home safety systems
- Home automation & remote controls
- Environment smart nodes

**Smart Industrial**

**Healthcare**
- Activity monitor
- Heart rate and ECG monitor
- Blood Pressure monitor

**Fitness & Wellness**
- Activity Monitor
- Smart watch / glass
- OHRM
- Smart Clothing

**Life augmented**
STM32 Nucleo Ecosystem
Building block approach

The building blocks
- Sense
- Connect
- Translate
- Move / actuate
- Power
- Process
- Software

Your need
- Data Collect
- Data Transmit
- Data Access
- Data Create
- Data Power
- Data Process

Our answer
STM32 Nucleo Development Boards

- Based on ST’s 32-bit ARM Cortex-M based STM32 microprocessors
- Development boards for all STM32 families available or planned

STM32 Nucleo Expansion Boards

- Boards with additional functionality: sensing, connectivity, power, analog
- Plugged on top or bottom of the STM32 Nucleo developer board or stacked on top of other expansion boards
- Leveraging ST wide product portfolio
STM32 Nucleo Ecosystem
Software Components

• **STM32Cube**
  - A set of free of charge tools and embedded Software bricks to enable fast and easy development on the STM32
    - PC-based STM32CubeMX for graphical configuration of the STM32
    - Hardware Abstraction Layer for easy porting from one STM32 device to another
    - Middleware bricks for the most common functions
    - Hundreds of code use examples are also included

• **STM32Cube Expansion Software**
  - Free of charge for every STM32 Nucleo expansion boards

• **Multiple Development Environments**
  - Compatible with a number of Development Environments including IAR EWARM, Keil MDK, mbed and GCC-based IDEs

• **Develop community and support**
  - Online communities, Development tools, documentation and user guides etc.
STM32CubeMX Configurator
STM32 Nucleo Expansion board examples

Bluetooth
Bluetooth Low Energy Expansion Board based on BlueNRG
Available Now

NFC
Dynamic NFC tag Expansion Board based on M24SR
Available Now

Stepper Motor Driver
Stepper motor driver expansion board based on easySPIN™ L6474
Available Now

Motion & Environmental
LSM6DS0 3-axis accelerometer + 3-axis gyroscope, the LIS3MDL 3-axis magnetometer, the HTS221 humidity sensor and the LPS25H pressure sensor.
Available Now
Flexible power supply through USB or external source

Integrated debugging and programming probe

STM32 microcontroller

Complete product range from ultra-low power to high-performance

Morpho and Arduino™ expansion headers
Sensor Expansion Board
X-NUCLEO-IKS01A1

Available now

DIL24 support for new devices i.e. LSM6DS3

Motion MEMS sensors
LIS3MDL LSM6DS0

Environmental sensors
LPS25H HTS221

www.st.com/mems
ST free Apps for MEMS

Unleashing great software and algorithms for innovative Motion MEMS applications. Bring your ideas to life.augmented now!

Open.MEMS MotionFX went live Nov.11, 2014

www.st.com/openmems
Open.MEMS flow

- Download and execute X-CUBE-MEMS2 from ST Web
- Follow the Wizard instructions
- Read and accept the license agreement to install the SW

- Run the license wizard

- Connect the Nucleo board and generate the sensor fusion license request
# STM32 Nucleo Expansion Boards

## Addressing the Functional Needs

<table>
<thead>
<tr>
<th>What do you want to do?</th>
<th>What you need</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sense</td>
<td>Motion sensors</td>
<td>ST (Q4 2014)</td>
</tr>
<tr>
<td></td>
<td>Environmental sensors</td>
<td>ST (Q4 2014)</td>
</tr>
<tr>
<td></td>
<td>Proximity sensors</td>
<td>ST (Q4 2014)</td>
</tr>
<tr>
<td></td>
<td>Microphone</td>
<td>ST (Q1 2015)</td>
</tr>
<tr>
<td>Connect</td>
<td>Bluetooth Low Energy</td>
<td>ST (available now)</td>
</tr>
<tr>
<td></td>
<td>Wi-Fi</td>
<td>ST (Q1 2015)</td>
</tr>
<tr>
<td></td>
<td>Sub-GHz radio</td>
<td>ST (Q4 2014)</td>
</tr>
<tr>
<td></td>
<td>NFC</td>
<td>ST (available now)</td>
</tr>
<tr>
<td></td>
<td>GNSS</td>
<td>ST (H1 2015)</td>
</tr>
<tr>
<td></td>
<td>Cellular</td>
<td>Third party</td>
</tr>
<tr>
<td></td>
<td>Ethernet</td>
<td>Third party</td>
</tr>
<tr>
<td>Move/actuate</td>
<td>Stepper motor driver</td>
<td>ST (available now)</td>
</tr>
<tr>
<td></td>
<td>DC &amp; BLDC motor driver</td>
<td>ST (Q1 2015)</td>
</tr>
<tr>
<td></td>
<td>Relay</td>
<td>Third party</td>
</tr>
<tr>
<td>Power</td>
<td>Energy management &amp; battery</td>
<td>ST (Q1 2015)</td>
</tr>
<tr>
<td>Translate</td>
<td>Audio amplifier</td>
<td>ST (Q4 2014)</td>
</tr>
<tr>
<td></td>
<td>OpAmp</td>
<td>ST (Q1 2015)</td>
</tr>
</tbody>
</table>
STM32Cube Expansion SW

Each Nucleo Expansion board leverages STM32Cube expansion SW that enables users to start coding their application from day one.

- Cube HAL pre-integrated drivers
- Specific middleware libraries
- Relevant application examples, with ready-made IDE projects
- Example of “vertical prototypes” integrating functionalities from several expansion boards/SW
- Released in source code with permissive license (with a few exceptions)
Serving the needs of vertical markets

<table>
<thead>
<tr>
<th>Functions</th>
<th>Vertical Market</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor Control</td>
<td>Appliance</td>
</tr>
<tr>
<td>Sensing</td>
<td>Robotics</td>
</tr>
<tr>
<td>Processing</td>
<td>Wearable</td>
</tr>
<tr>
<td>Power Management</td>
<td>Building Automation</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Automotive</td>
</tr>
<tr>
<td>Power conversion</td>
<td></td>
</tr>
<tr>
<td>Lighting</td>
<td></td>
</tr>
<tr>
<td>Positioning</td>
<td></td>
</tr>
</tbody>
</table>

- Key Function in the application
- Emerging Function in the application
Lowering the Barriers for Developers

Easy Access to technology

Idea

Rapid Device & SW Development

Closer to final Form factor Device

Field Test

Final Device Form factor

Production SW

Fast, flexible, affordable and based on commercial components
Ready to Use Solutions for Vertical Markets

Rapid Prototyping with STM32 Nucleo Development Ecosystem

- Modular hardware enables broad deployment through a standardized development framework
- Stack multiple expansion boards to add power management, sensors, connectivity and more to the STM32 Nucleo development boards
- Intuitive software tools offer code examples and documentations to get up and running quickly
- Price competitive boards

A wide offer of evaluation boards (STEVAL) to address Vertical Markets

- Automation 9%
- Lighting 13%
- Sensors (MEMS) 16%
- Motor Control 6%
- Power Management 26%
- Home Consumer 12%
- Others 18%

>410 Evaluation Boards
Takeaways

Big opportunity as electronics penetrate new sectors with the IoT

Little electronics skills
Need easy access
All the key components are already here

ST has a solution to make access to electronics easy

Easy
Affordable
STM32 Nucleo Ecosystem

Fast
Open licenses
Commercial grade components
STM32Cube
STM32Cube FW solution presentation with focus on HAL and STM32CubeMx

Slim Jallouli
slim.jallouli@st.com
STM32Cube™ Introduction

• STM32Cube™ includes:
  • A configuration tool, STM32CubeMX generating initialization code from user choices
  • A full embedded software offer, delivered per series (like STM32CubeF4) with:
    • An STM32 Abstraction Layer embedded software: STM32Cube HAL
    • A consistent set of Middlewares: RTOS, USB, TCP/IP, Graphics, …
Link for STM32CubeF4

STM32Cube FW package presentation
STM32Cube FW package block view

Evaluation boards

Discovery boards

Nucleo boards

Board Demonstrations

Middleware level examples

Networking
LwIP TCP/IP & Polar SSL

USB
Host & Device

Graphics
STemWin

File system
FATFS

RTOS
FreeRTOS

Middleware

HAL level examples

Hardware Abstraction Layer API

Boards Support Packages

Drivers

Utilities

CMSIS

F4 Family

STM32F401

STM32F405/7

STM32F429

STM32F439

STM32F401

STM32F405/7

STM32F429

STM32F439

STM32Cube Technical Update (Part 1)
STM32Cube projects folder organization

- Projects
  - STM324x9L_EVAL
  - STM324xG_EVAL
    - Applications
      - Camera
      - Display
      - FatFs
      - FreeRTOS
      - LwIP
      - PolarSSL
      - STMWin
      - USB_Device
      - USB_Host
  - Demonstrations
    - Examples
      - ADC
      - BSP
      - CAN
      - Cortex
      - CRC
      - CRYP
      - DAC
      - DCMI
      - DMA
      - UART
      - WWDG
      - Templates
        - EWARM
        - Inc
        - MDK-ARM
        - Src
        - TrueSTUDIO
          - readme.txt
      - STM32F4-Discovery
      - STM32F401-Discovery
      - STM32F429-Discovery
      - STM32F4xx-Nucleo
STM32Cube versioning and maintenance

- **STM32Cube_FW_[Product family]_VX.Y.Z**
  - X: STM32Cube version: V1 « Consistent » (next to come: V2 « Integrated »)
  - Y: major enhancements and/or bug fixes
  - Z: minor enhancements and/or bug fixes

- Components of the STM32Cube have their own version number which can be found in the release note document available with each component

- The STM32Cube FW package will be maintained regularly through
  - Full release
  - Patch release
    - Patches allow to fix or enhance an STM32Cube component (or a set of components)
    - The patch contains new release of the component(s)
    - Any released patch will include all previous patches relative to current STM32Cube release

- The updater tool available with STM32CubeMX PC tool allows automatic notification and download of new STM32Cube release or patch
STM32CubeF4 Documentation Status

- UM1730 Getting Started with STM32Cube Firmware package for STM32F4xx series ➔ Available on the web
- UM1721 Developing Applications on STM32Cube with FatFs ➔ Available on the web
- UM1722 Developing Applications on STM32Cube with RTOS ➔ Available on the web
- UM1725 HAL Driver ➔ Will be available on the web 14W14
- UM1734 STM32Cube USB Device library ➔ Will be available on the web 14W14
- UM1720 STM32Cube USB Host library ➔ Will be available on the web 14W15
- UM1723 STM32Cube PolarSSL example ➔ Will be available on the web 14W13
- UM1709 STM32Cube Ethernet IAP example ➔ Will be available on the web 14W13
- UM1713 STM32Cube interfacing with LwIP and examples ➔ Will be available on the web 14W13
- UMxxxx How to migrate an STM32 Application from StdLib to STM32Cube ➔ Will be available on the web 14W14
- UMxxxx STM32CubeF4 Demonstration ➔ Will be available on the web 14W14
STM32Cube Hardware Abstraction Layer (HAL)
HAL overview
Introduction to HAL

• The STM32Cube Hardware abstraction layer (HAL) replaces the standard peripheral library

• The hardware abstraction allows to offer
  • User friendly and portable APIs that hides the HW complexity
  • An easy and quick migration of user’s application from a product family to another

• The HAL covers all product peripherals including advanced peripherals like USB, Ethernet,…

• The HAL comes with an extensive set of examples running on available boards (evalboard, discovery, Nucleo) with ready projects for three toolsets (IAR, Keil, Attotic)

• All HAL drivers passed CodeSonar C code verification tool
HAL overview
HAL APIs main features

• Cross-family portable API set for the common peripheral features and extension APIs in cases of specific peripheral features

• HAL drivers support three API programming models: polling, interrupt, and DMA

• APIs are RTOS compliant
  • Fully reentrant APIs
  • Systematic usage of timeouts when doing polling

• Peripheral multi-instance support
  • Allows concurrent API call for different instance of a peripheral (USART1, USART2,...)

• HAL APIs implement user callback functions mechanism
  • Peripheral Init/DeInit HAL APIs call user callback function to do peripheral system level initialization/DeInitialization (clock, GPIOs, interrupt, DMA)
  • Peripherals interrupt events
  • Error events
### HAL file components

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stm32f4xx_hal_ppp.c/.h</td>
<td>peripheral driver with portable APIs</td>
</tr>
<tr>
<td>stm32f4xx_hal_ppp_ex.c/.h</td>
<td>extended peripheral features APIs</td>
</tr>
<tr>
<td>stm32f4xx_hal.c</td>
<td>contains HAL common APIs (HAL_Init, HAL_DeInit, HAL_Delay,…</td>
</tr>
<tr>
<td>stm32f4xx_hal.h</td>
<td>HAL header file, it should be included in user code</td>
</tr>
<tr>
<td>stm32f4xx_hal_conf.h</td>
<td>config file for HAL, should be customized by user to select the peripherals to be included</td>
</tr>
<tr>
<td>stm32f4xx_hal_def.h</td>
<td>contains HAL common typedefs and macros</td>
</tr>
<tr>
<td>stm32f4xx_ll_ppp.c</td>
<td>implements low level functions in case of some complex peripherals, they are called from stm32f4xx_hal_ppp.c</td>
</tr>
</tbody>
</table>
HAL overview
HAL inclusion in user application

User application

- user_code.c/.h
- stm32f4xx_it.c/.h
- stm32f4xx_hal_conf.h
- stm32f4xx_hal_ppp.c
- stm32f4xx_hal_ppp_ex.c
- stm32f4xx_hal.c

HAL includes

- stm32f4xx_hal.h
- stm32f4xx_hal_ppp.h
- stm32f4xx_hal_ppp_ex.h
- stm32f4xx_hal_def.h

HAL

CMSIS

- system_stm32f4xx.c
- startup_stm32f4xx.s
- stm32f4xx.h

Startup

- startup_stm32f4xx.s

System

- system_stm32f4xx.c
HAL common APIs

• Implemented in file stm32f4x_hal.c, main APIs are

  • HAL_Init(), need to be called at application startup
    • Initializes data/instruction cache and pre-fetch queue
    • Sets Systick to generate interrupt each 1ms (based on HSI clock) with lowest priority
    • Sets priority grouping to 4 preemption bits
    • Calls function HAL_MspInit() which a is user callback function to do system level initializations (clocks, gpios, DMA, interrupts). HAL_MspInit() is defined as “weak” empty function in HAL

  • HAL_DeInit()
    • Resets all peripherals
    • Calls function HAL_MspDeInit() which a is user callback function to do system level De-Initializations

  • HAL_GetTick()
    • Get current tick counter (incremented in systick interrupt)
    • Used by peripherals drivers to handle timeouts

  • HAL_Delay()
    • Implements a delay in ms (using systick)

• Note: in some particular cases systick interrupt priority need to be changed in order to handle delay management inside peripherals interrupt handlers/callbacks
HAL system peripherals
HAL system peripherals
RCC HAL driver

- Two main functions for clock configuration
  - HAL_RCC_OscConfig (RCC_OscInitTypeDef *RCC_OscInitStruct)
    - Configures/Enables multiple clock sources (HSE, HSI, LSE, LSI, PLL)
  - HAL_RCC_ClockConfig (RCC_ClkInitTypeDef *RCC_ClkInitStruct, uint32_t FLatency)
    - Selects system clock source
    - Configures AHB, APB1 and APB2 clock dividers
    - Configures Flash wait states
    - Updates systick config following HCLK clock changes

- Other functions in RCC HAL driver include
  - Clock de-init function HAL_RCC_DeInit() : allows to return to reset state clock configuration
  - Get clock functions to get various clock configs (system clock, HCLK, PCLK1, PCLK2, …)
  - MCO config function

- A set of macros are defined in stm32f4xx_hal_rcc.h
  - Allow elementary operations on RCC block registers like for example peripherals clock gating/reset control
    - Peripheral clock enable/disable: __SPI1_CLK_ENABLE() / __DISABLE()
    - Peripheral reset control : __SPI_FORCE_RESET() / __RELEASE_RESET()
**HAL system peripherals**

**GPIO HAL driver**

- **GPIO HAL APIs are**
  - HAL_GPIO_Init() / HAL_GPIO_DeInit()
  - HAL_GPIO_ReadPin() / HAL_GPIO_WritePin()
  - HAL_GPIO_TogglePin()

- **In addition to standard modes for GPIO (input, output, analog), pin mode can be configured as EXTI with interrupt or event generation**
  - When selecting EXTI mode with interrupt generation, user need to call HAL_GPIO_EXTI_IRQHandler() from stm32f4xx_it.c file and implement callback function HAL_GPIO_EXTI_Callback()

- **GPIO_InitTypeDef stucture**

```c
/*
 * @brief GPIO Init structure definition
 *
 typedef struct
 {
   uint32_t Pin;  /*!< Specifies the GPIO pins to be configured. 
                   This parameter can be any value of @ref GPIO_pins_define */
   uint32_t Mode; /*!< Specifies the operating mode for the selected pins. 
                   This parameter can be a value of @ref GPIO_mode_define */
   uint32_t Pull;  /*!< Specifies the Pull-up or Pull-Down activation for the selected pins. 
                   This parameter can be a value of @ref GPIO_pull_define */
   uint32_t Speed; /*!< Specifies the speed for the selected pins. 
                   This parameter can be a value of @ref GPIO_speed_define */
   uint32_t Alternate;  /*!< Peripheral to be connected to the selected pins 
                        This parameter can be a value of @ref GPIO_Alternat_function_selection */
 }GPIO_InitTypeDef;
```
## DMA HAL APIs

<table>
<thead>
<tr>
<th>DMA HAL APIs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_DMA_Init</td>
<td>Initializes a DMA channel</td>
</tr>
<tr>
<td>HAL_DMA_DeInit</td>
<td>De-initializes a DMA channel</td>
</tr>
<tr>
<td>HAL_DMA_Start</td>
<td>Starts DMA channel</td>
</tr>
<tr>
<td>HAL_DMA_Start_IT</td>
<td>Starts DMA channel with interrupt generation at end of transfer or half transfer or on DMA error</td>
</tr>
<tr>
<td>HAL_DMA_Abort</td>
<td>Aborts a DMA transfer</td>
</tr>
<tr>
<td>HAL_DMA_PollForTransfer</td>
<td>Blocking function that polls for transfer complete or half complete, this function can also return a Timeout or a DMA error</td>
</tr>
<tr>
<td>HAL_DMA_IRQHandler</td>
<td>Interrupt handler for DMA</td>
</tr>
<tr>
<td>HAL_DMA_GetState</td>
<td>Gets DMA channel state</td>
</tr>
<tr>
<td>HAL_DMA_GetError</td>
<td>Gets DMA error code</td>
</tr>
</tbody>
</table>
HAL system peripherals
Cortex HAL driver

• Cortex HAL driver provides APIs for handling NVIC and Systick, supported APIs include
  • HAL_NVIC_SetPriorityGrouping
  • HAL_NVIC_SetPriority
  • HAL_NVIC_EnableIRQ /HAL_NVIC_DisableIRQ
  • HAL_SYSTICK_Config
  • HAL_SYSTICK_CLKSourceConfig
HAL system peripherals
PWR HAL driver

• PWR HAL driver handles power management features
  • PVD configuration, enabling/disabling and interrupt handling
    • HAL_PWR_PVDConfig()
    • HAL_PWR_EnablePVD() / HAL_PWR_DisablePVD()
    • HAL_PWR_PVD_IRQHandler()
    • HAL_PWR_PVDCallback()
  • Low power mode entry
    • HAL_PWR_EnterSLEEPMode()
    • HAL_PWR_EnterSTOPMode()
    • HAL_PWR_EnterSTANDBYMode()
• Extension function are available, these are
  • Flash overdrive control and flash power-down (for F429/F439 only)
    • HAL_PWREx_ActivateOverDrive()
    • HAL_PWREx_EnableFlashPowerDown()
  • Backup domain registers enable/disable
    • HAL_PWREx_EnableBkUpReg() / HAL_PWREx_DisableBkUpReg
Peripherals HAL driver model
Peripheral HAL driver model
HAL peripheral Handle

• A handle structure is allocated for each instance of a peripheral

• The handle structure allows to save particular parameters for a peripheral instance (peripherals current config/initiation parameters, global variables, peripheral state, DMA channel handles)

• The handle structure is peripheral dependent, the following is an example for the USART handle (members in green should be initialized before calling function HAL_PPP_Init()):

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>USART_TypeDef*</td>
<td>Pointer to the register base address</td>
</tr>
<tr>
<td>Init</td>
<td>USART_InitTypeDef</td>
<td>USART communication initialization parameters: will be initialized when calling HAL_USART_Init()</td>
</tr>
<tr>
<td>pTxBuffPtr</td>
<td>uint8_t*</td>
<td>Pointer to Transmit buffer</td>
</tr>
<tr>
<td>pRxBuffPtr</td>
<td>uint8_t*</td>
<td>Pointer to Receive buffer</td>
</tr>
<tr>
<td>TxXferSize</td>
<td>uint16_t</td>
<td>Usart Tx Transfer size</td>
</tr>
<tr>
<td>RxXferSize</td>
<td>uint16_t</td>
<td>Usart Rx Transfer size</td>
</tr>
<tr>
<td>TxXferCount</td>
<td>uint16_t</td>
<td>counter of the transmitted data</td>
</tr>
<tr>
<td>RxXferCount</td>
<td>uint16_t</td>
<td>counter of the received data</td>
</tr>
<tr>
<td>Lock</td>
<td>HAL_LockTypeDef</td>
<td>Lock object: used internally</td>
</tr>
<tr>
<td>State</td>
<td>USART_StateTypeDef</td>
<td>USART peripheral state</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>unit8_t</td>
<td>Error code</td>
</tr>
<tr>
<td>hdmatx</td>
<td>DMA_HandleTypeDef*</td>
<td>DMA handle for Tx: should be initialized in case DMA will be used for USART transmit operation</td>
</tr>
<tr>
<td>hdmarx</td>
<td>DMA_HandleTypeDef*</td>
<td>DMA handle for Rx: should be initialized in case DMA will be used for USART receive operation</td>
</tr>
</tbody>
</table>
Peripheral HAL driver model
Driver API groups

- Peripheral drivers APIs are organized in four groups
  - Initialization and de-initialization functions
  - I/O operation functions
  - Peripheral control functions
  - Peripheral State and Errors functions

<table>
<thead>
<tr>
<th>API group</th>
<th>examples</th>
</tr>
</thead>
</table>
| Initialization and de-initialization | HAL_USART_Init()  
                              | HAL_USART_DeInit()                           |
| I/O operation (or process)       | HAL_SPI_Receive()  
                              | HAL_USART_Transmit_DMA()                    |
| Peripheral control               | HAL_ADC_ConfigChannel()                      |
|                                  | HAL_TIM_OC_ConfigChannel()                   |
| Peripheral state and error       | HAL_I2C_GetState()                           |
|                                  | HAL_I2C_GetError()                           |
Peripheral HAL driver model
Interrupt handler & callback functions

• Besides the APIs, HAL peripheral drivers implement
  • The peripheral interrupt handler: should be called from stm32f4xx_it.c
  • User callback functions

• User callback functions are defined as empty functions with “weak” attribute they need to be redefined in user code

• Three types of user callbacks functions are defined
  • Peripheral system level initialization/de-Initialization callbacks: HAL_PPP_MspInit()/_DeInit
  • Process complete callbacks: HAL_PPP_ProcessCpltCallback
  • Error callback: HAL_PPP_ErrorCallback

<table>
<thead>
<tr>
<th>Callback functions</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_PPP_MspInit() / _DeInit()</td>
<td>Ex: HAL_USART_MspInit() Called from HAL_PPP_Init() API function to do peripheral system level initialization (GPIOs, clock, DMA, interrupt)</td>
</tr>
<tr>
<td>HAL_PPP_ProcessCpltCallback</td>
<td>Ex: HAL_USART_TxCpltCallback Called by peripheral or DMA interrupt handler on process complete</td>
</tr>
<tr>
<td>HAL_PPP_ErrorCallback</td>
<td>Ex: HAL_USART_ErrorCallback Called by peripheral or DMA interrupt handler on error occurrence</td>
</tr>
</tbody>
</table>
Peripheral HAL driver model

Process API types

• **Blocking polling process APIs**
  • blocks until end of process, timeout or error
  • Ex: HAL_USART_Receive()

• **Non blocking process APIs**
  • Case of Start APIs: exits directly after starting the process
  • Ex: HAL_ADC_Start()

• **Non blocking process APIs with peripheral interrupt generation at end of process**
  • User notification of end of process or error through user callback functions
  • Ex: HAL_USART_Receive_IT()

• **Non blocking APIs with DMA transfer and DMA interrupt generation at end of transfer**
  • User notification of end of process or error through user callback functions
  • Ex: HAL_USART_Receive_DMA()

• **Note**: user callbacks are the same in case of peripheral or interrupt DMA
  • Ex: HAL_USART_RxCpltCallback() is called from both peripheral and DMA interrupt handlers
STM32 Nucleo Board
STM32F072 Nucleo Board

- STM32 MCU with 64 pins
- Integrated ST-Link/V2-1: mass storage device flash programming + USB Virtual Com Port
- 2 push buttons, 2 color LEDs
- Arduino extension connectors: easy access to add-ons
- Flexible board power supply: through USB or external source
- STM32 MCU with 64 pins
- Morpho extension headers: direct access to all MCU I/Os

www.st.com/stm32nucleo
Arduino Compatible Header
Morpho Headers

STM32Cube Technical Update (Part 1)
STM32CubeMX Demo
Demo and Presentation of mbed
STM32F072 Nucleo Board

- STM32 MCU with 64 pins
- Integrated ST-Link/V2-1: mass storage device flash programming + USB Virtual Com Port
- 2 push buttons, 2 color LEDs
- Arduino extension connectors: easy access to add-ons
- STM32 MCU with 64 pins
- Morpho extension headers: direct access to all MCU I/Os

Flexible board power supply: through USB or external source

www.st.com/stm32nucleo
The Development Environment

- This innovative rapid development environment makes it easy for you to quickly create your connected device

STM32 Nucleo board

USB cable

Laptop

www.mbed.org
5 Steps to achieve a connected device

1. You need a Nucleo board and USB cable

2. Go to: http://www.mbed.org and connect to the ST Nucleo platform

3. Add a connectivity module

4. Find and attach sensors

5. Connect your device to the cloud (https://m2x.att.com), phone or other device
Connectivity and Sensor Options

• Sensor and I/O Options

- Bluetooth LE
- LoRa Radio
- WiFi
- NFC Radio
- Cellular Modem
- mbed ready connectivity modules

- Button - Joystick
- Vibration Motor
- Gas Sensor
- Buzzer
- Barometer
- Display
- Gyro - Accel - Mag
- Temp - Humidity - Press
- Heart Rate
Thank you
CONFIDENTIALITY OBLIGATIONS:

THIS DOCUMENT CONTAINS SENSITIVE INFORMATION.

IT IS CLASSIFIED "MICROCONTROLLERS, MEMORIES & SECURE MCUs (MMS) RESTRICTED AND ITS DISTRIBUTION IS SUBMITTED TO ST/MMS AUTHORIZATION

AT ALL TIMES YOU SHOULD COMPLY WITH THE FOLLOWING SECURITY RULES:
- DO NOT COPY OR REPRODUCE ALL OR PART OF THIS DOCUMENT
- KEEP THIS DOCUMENT LOCKED AWAY
- FURTHER COPIES CAN BE PROVIDED ON A "NEED TO KNOW BASIS", PLEASE CONTACT YOUR LOCAL ST SALES OFFICE OR DOCUMENT WRITER.

Information furnished is believed to be accurate and reliable. However, STMicroelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of STMicroelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. STMicroelectronics products are not authorized for use as critical components in life support devices or systems without express written approval of STMicroelectronics.

The ST logo is registered trademark of STMicroelectronics
All other names are the property of their respective owners

© 2010 STMicroelectronics - All Rights Reserved

STMicroelectronics group of companies Australia - Brazil - Canada - China - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States.

www.st.com