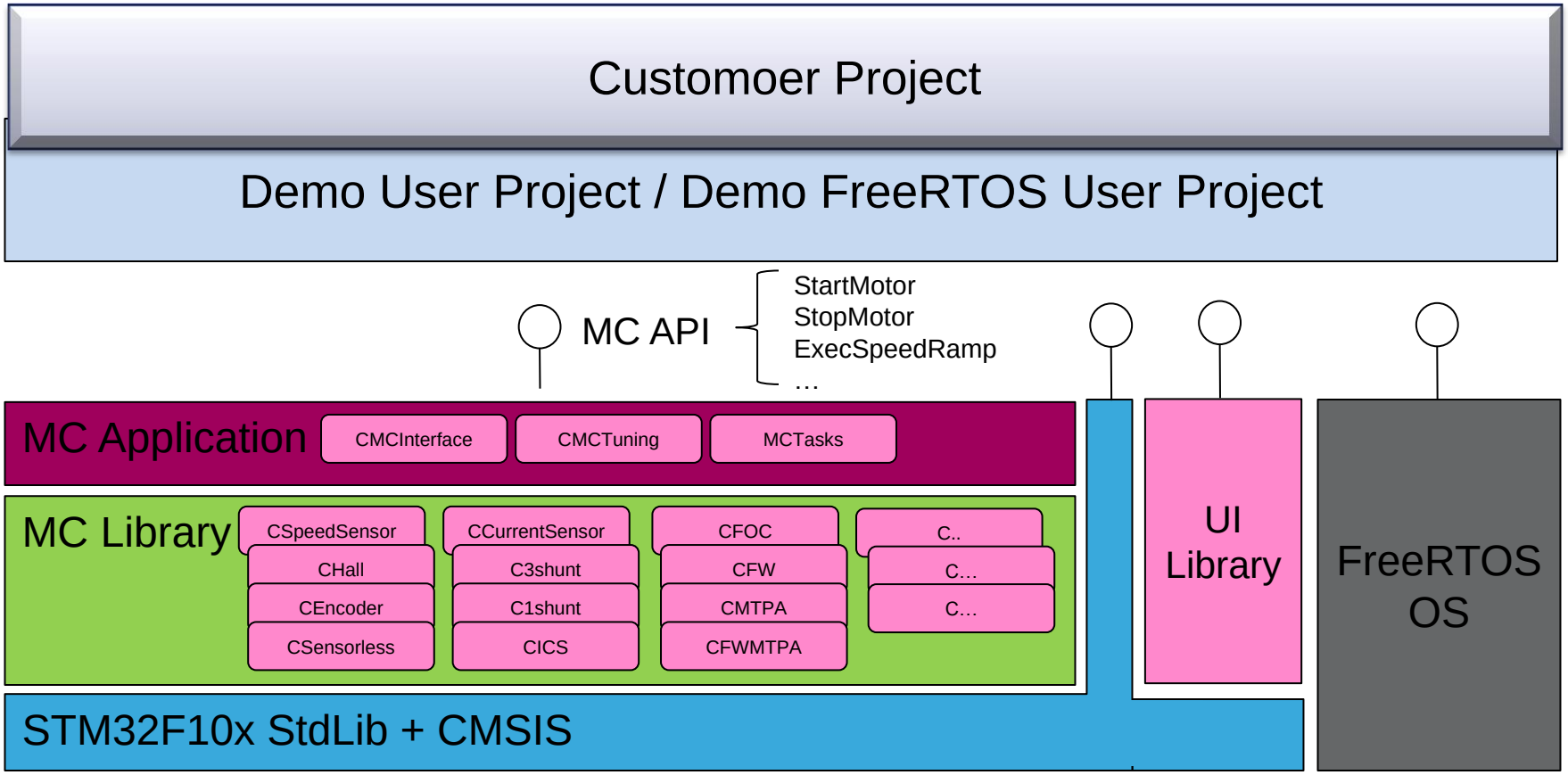
Agenda

- 1st day – Afternoon
  - MC Application
    - Interface
    - Tuning
    - Tasks
    - Classes interaction
      - Current regulation
      - Ramp-up
      - Encoder alignment
  - Speed sensors updates:
    - Sensorless algorithm improvement
  - How to create User Project Interacting with MC Application
  - Dual motor control
    - Resources sharing
    - Supported configurations
    - Code size efficiency
  - Current reading sensor update

Integration with MC Application

- This section explains how to integrate the MC Application with an User Project



Configuring the project

- STEP 1
  - user project should include sources:
    - $\Libraries\CMSIS\CM3\CoreSupport\core_cm3.c
    - $\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x\system_stm32f10x.c
    - $\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x\startup\XXX\startup_stm32f10x_YYY.s
      (XXX according to IDE) (YYY according to device)
    - $\Project\stm32f10x_it.c (removing conditional compilation, can bemodified)
    - $\Project\stm32f10x_MC_it.c (GUI generated according to params)
  - standard peripheral driver sources as needed from
    - $\Libraries\STM32F10x_StdPeriph_Driver\src\

Configuring the project

- STEP 2
  - path inclusion:
    - $\Libraries\CMSIS\CM3\CoreSupport\
    - $\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x\
    - $\Libraries\STM32F10x_StdPeriph_Driver\inc\
    - $\MC library\interface\common\
    - $\MC Application\interface\
    - $\System & Drive Params\
  - linker inclusion:
    - (if in single motor drive)
      $*\MC Library Compiled\Exe\MC_Library_single_drive.a
    - (if in dual motor drive)
      $*\MC Library Compiled\Exe\MC_Library_dual_drive.a
    - $**\MC Application Compiled\Exe\MC Application.a

Configuring the project

- STEP 3
  - define symbols:
    - USE_STDPERIPH_DRIVER
    - STM32F10X_MD \ STM32F10X_HD \ STM32F10X_MD_VL …
- STEP 4
  - header files inclusion in sources that interact with MC API
    - #include "MCTuningClass.h“
    - #include "MCInterfaceClass.h“
    - #include "MCTasks.h“
  - header file inclusion to read some parameter from #defines
    - #include "Parameters conversion.h“
    - #include "Parameters conversion motor 2.h“

Configuring the application

- Set the STM32 NVIC (Nested Vectored Interrupt Controller) priority group configuration (the default option is NVIC_PriorityGroup_3). The alternative option, left to user choice, is NVIC_PriorityGroup_2:

```
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_3);
```

- Priorities used in the MC Library:

IRQ	pre-emption
TIM1 UPDATE	0
TIM8 UPDATE (F103HD/XL only)	0
DMA	0
ADC1_2 (F103 only)	1
ADC3 (F103HD/XL only)	1
ADC1 (F100 only)	1
USART (UI library)	2
TIMx GLOBAL (speed sensor decoding)	2

Timebase, clocks the MCA needs

- A timebase is needed to clock the MC Application; the demo timebase.c can be considered an example or used as it is; resources it uses are
  - Systick timer
  - SysTick_Handler, PendSV_Handler
- The timebase should provide these clocks:

No.	Function to call	Periodicity	Priority	Preemptiveness
*1	TSK_LowFrequencyTask	10ms	Base	Yes, over non MC functions.
*2	TSK_MediumFrequencyTask	Equal to that set in ST MC GUI, speed regulation execution rate	Higher then *1	Yes, over *1
*3	TSK_SafetyTask	0.5ms	Higher then *2	Yes, over *1, (optional over *2)

Priorities configuration, overall

- Non FreeRTOS

COMPONENT	Pre-emption priority
MC LIBRARY	0,1,2
TIMEBASE (MCA clocks)	3,4
USER	5,6,7

Priorities configuration, overall

- FreeRTOS

COMPONENT	Pre-emption priority	
MC LIBRARY	0,1,2 (3 reserved)	
USER (only FreeRTOS API!)	4,5,6	
FreeRTOS	7	RTOS priority
	MCA clock tasks	Highest
	User Tasks	Lower

MC Application bootstrap

- from a source file that includes MC API:
  - Declare a static array of type CMCI (MC Interface class)
    - CMCI oMCI[MC_NUM]; /* MC_NUM is the number of motors to drive*/
  - Declare a static array of type CMCT (MC Tuning class)
    - CMCT oMCT[MC_NUM]; /* MC_NUM is the number of motors to drive*/
  - Start the MC Interface boot process:
    - MCboot(oMCI,oMCT);

and.. send commands to the MC API!

- It's now possible to send command to the MC API, for instance :
  - Reference speed modification, it should be done before starting the motor:
    - MCI_ExecSpeedRamp(oMCI[i],100,1000);
  - Start/Stop motor
    - MCI_StartMotor(oMCI[i]);
    - MCI_StopMotor(oMCI[i]);
  - Get the state of motor
    - MCI_GetSTMState(oMCI[i]);
  - FAULT status management
    - Get the information about both faults currently present(MSB 16-bit) and faults historically occurred(LSB 16-bit), fault bit difinition is in "MC_type.h":
      - STM_GetFaultState(MCT_GetStateMachine(oMCT[i]));
    - Acknowledge the FAULT status
      - MCI_FaultAcknowledged(oMCI[i]);
      - It return TRUE and move into STOP_IDLE If fault is over, else return FALSE