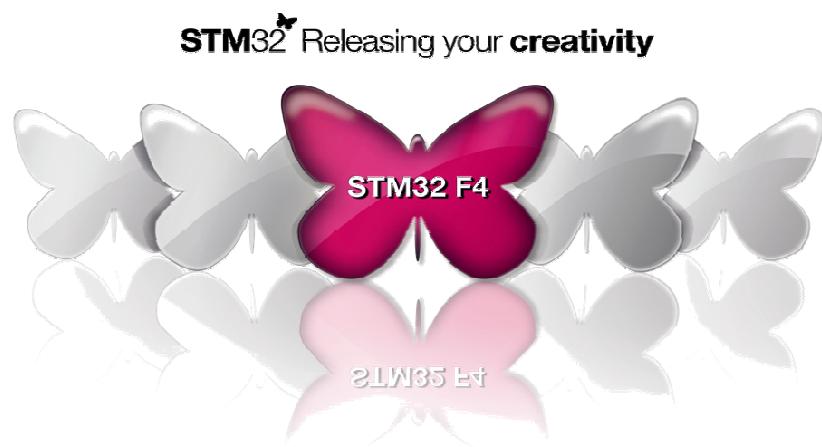


STM32F4 family – practical session

STM32F4 discovery kit



STM32F4_Discovery – in zoom



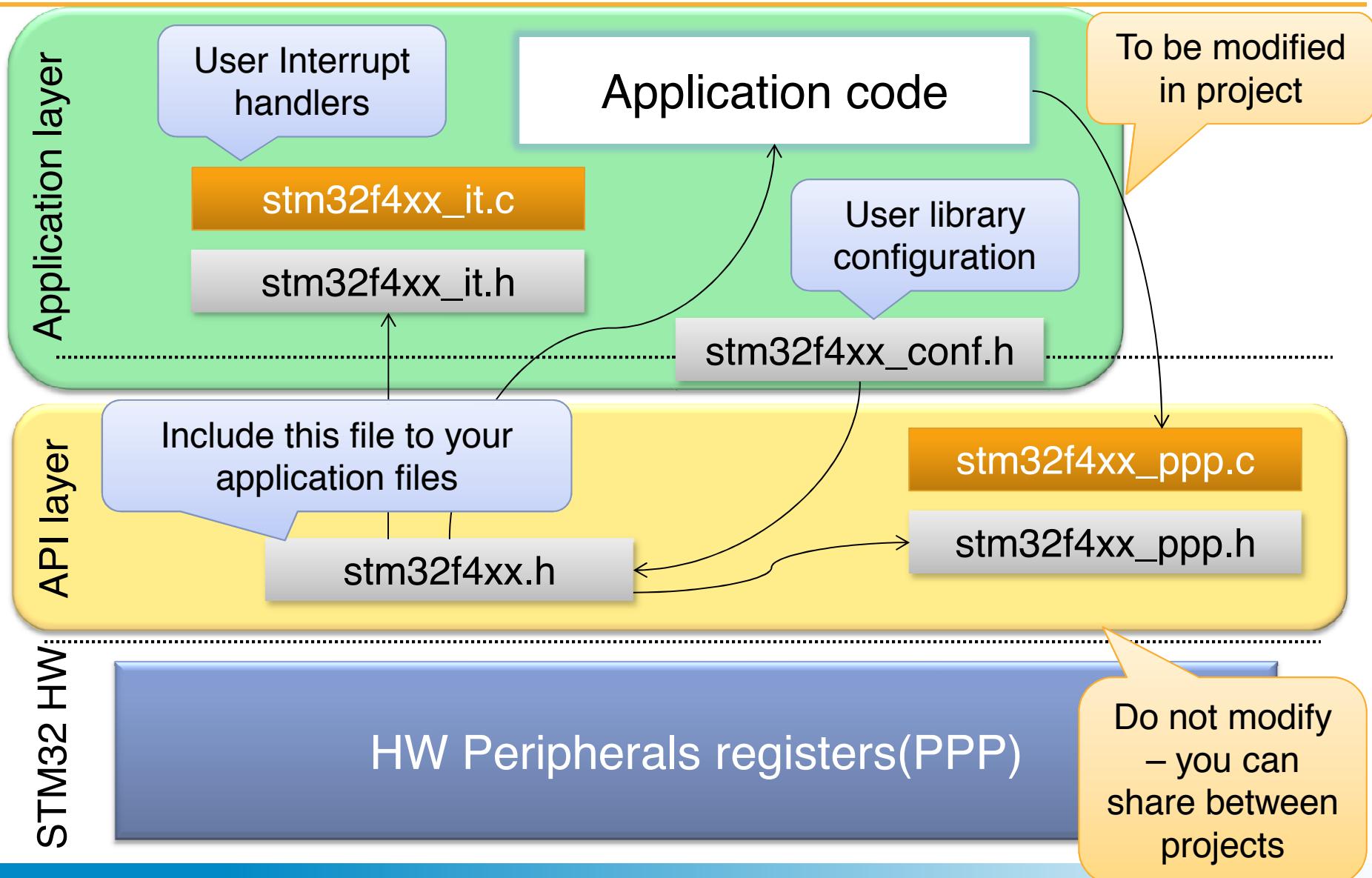
- STM32F4-Discovery board has two parts: ST-Link programmer/debugger and evaluation board with STM32F4xx MCU (CortexM4 based STM32)
ST-Link can be used either as on board programmer (SEL jumpers must be ON) or as standalone programmer for external STM32 (SEL jumpers must be OFF). It is working only in SWD mode.
- Evaluation part is equipped with **STM32F407VGT6** MCU (1MB Flash, 192kB RAM, 100pin package) with built in clocking system (main oscillator for HSE generator 8MHz).
it is possible to measure **current consumption** of the MCU using JP1 jumper.
- On board there are:
 - MEMS **accelerometer** (LIS302DL) connected via SPI1
 - Simple **user interface** (button + 4 LEDs)
 - Audio codec** with audio output
 - MEMS **microphone** (MP45DT02)
 - USB OTG** connector with 2 signaling LEDs

STM32 – startup procedure

- Minimum requirement for STM32 to start is to fill two first words in its vector table:
 - First word is always initial Main Stack Pointer value
 - Second word is always address of reset procedure
- It is recommended to implement as well main fault vectors (HardFault at least)
- In STM32 std library implementation with CMSIS standard, vector table is defined in startup file, which is prepared for each family member and each toolchain.
- In case of STM32F407VGT6 and Keil uVision it is **startup_stm32f4xx.s** file located in the folder **/Libraries/Device/STM32F4xx/Include** inside the project
- In ST library there are some additional operations put before **main()** function will be executed. The most important is **SystemInit()** function coming from **system_stm32f4xx.c** file. This function is doing configuration of clock system and some GPIO pins in order to cooperate with external components of the MCU. This is not necessary for standard application running.
- To **switch off** this procedure, lines concerning SystemInit function call in **startup_stm32f4xx.s** file should be **commented** (lines 178,179 in startup file)



STM32 FW library – API structure



STM32 – standard peripherals library



- STM32 standard peripherals library is written in ANSI C
- For each peripheral there are separate source and header files, i.e.:
 - `stm32f4xx_gpio.c`
 - `stm32f4xx_gpio.h`
- To use it, it is required to:
 - **#include "stm32f4xx.h"**
 - add to the project source files for used peripherals, i.e. `stm32f4xx_gpio.c` for GPIO
- In **`stm32f4xx_conf.h`** uncomment lines with peripherals you are using in applications, i.e.:
 - `#include "stm32f4xx_gpio.h"`
- Empty interrupt procedures are present in **`stm32f4xx_it.c`** file. All interrupt functions should be put there.
- **Interrupt function** do not require any special coding and are **void function(void)** type
- Whole manual for the library is available in html format (delivered with library package).
- Most of the peripherals has predefined one or two **data structures** which are used for the configuration. After fill up the structure it is used in **PPP_Init()** functions to configure registers in the peripherals

STM32 – library – how to use it ?

- Function and constant for each peripheral has prefix with its name, like: GPIO, TIM1:
ie. **GPIO_Init()**, **ADC_Channel_0**, **USART_IT_TXE**
- Most of the settings is in **1fromN** convention and allow to use concatenation, like:
GPIO_Pin_0 | GPIO_Pin_1, what means that **pins 0 and 1** from will be configured in the same time
- There are predefined types in **stm32f4xx.h** file, like:
 - **u8** – unsigned char
 - **u16** – unsigned short
 - **RESET / SET**
 - **FALSE / TRUE**
 - **DISABLE / ENABLE**
- Most of the peripherals (PPP) has set of instruction:
 - **PPP_DeInit(...)** – set all PPP register to its reset state
 - **PPP_Init(...)** – validation of the configuration for the peripheral
 - **PPP_Cmd(ENABLE/DISABLE)** – turn on/off PPP peripheral (not affects its clock)
 - **PPP_ITConfig(...)** – configuration (on/off) of sources of interrupts for PPP peripheral
 - **PPP_GetFlagStatus(...)** – read flags from the peripheral (polling)
 - **PPP_ClearFlag(...)** – clear flags from the peripheral
 - **PPP_ClearITPendingBit(...)** – clear IRQ flag

STM32 – library - FAQ

1. Compiler is reporting a lot of errors like:

Missing prototype

GPIO_Pin_0 undefined

Solution

Please check whether in ***stm32f4xx_conf.h*** all used library modules are **uncommented**

Please check, whether **USE_STDPERIPH_DRIVER** constant is defined in your environment

2. Linker is reporting a lot of errors like:

Lab_library.lkf:1 symbol _GPIO_WriteHigh not defined (Debug/main.o)

Solution

Please check whether all **library source files are added**, ***stm32f4xx_gpio.c*** in this case.

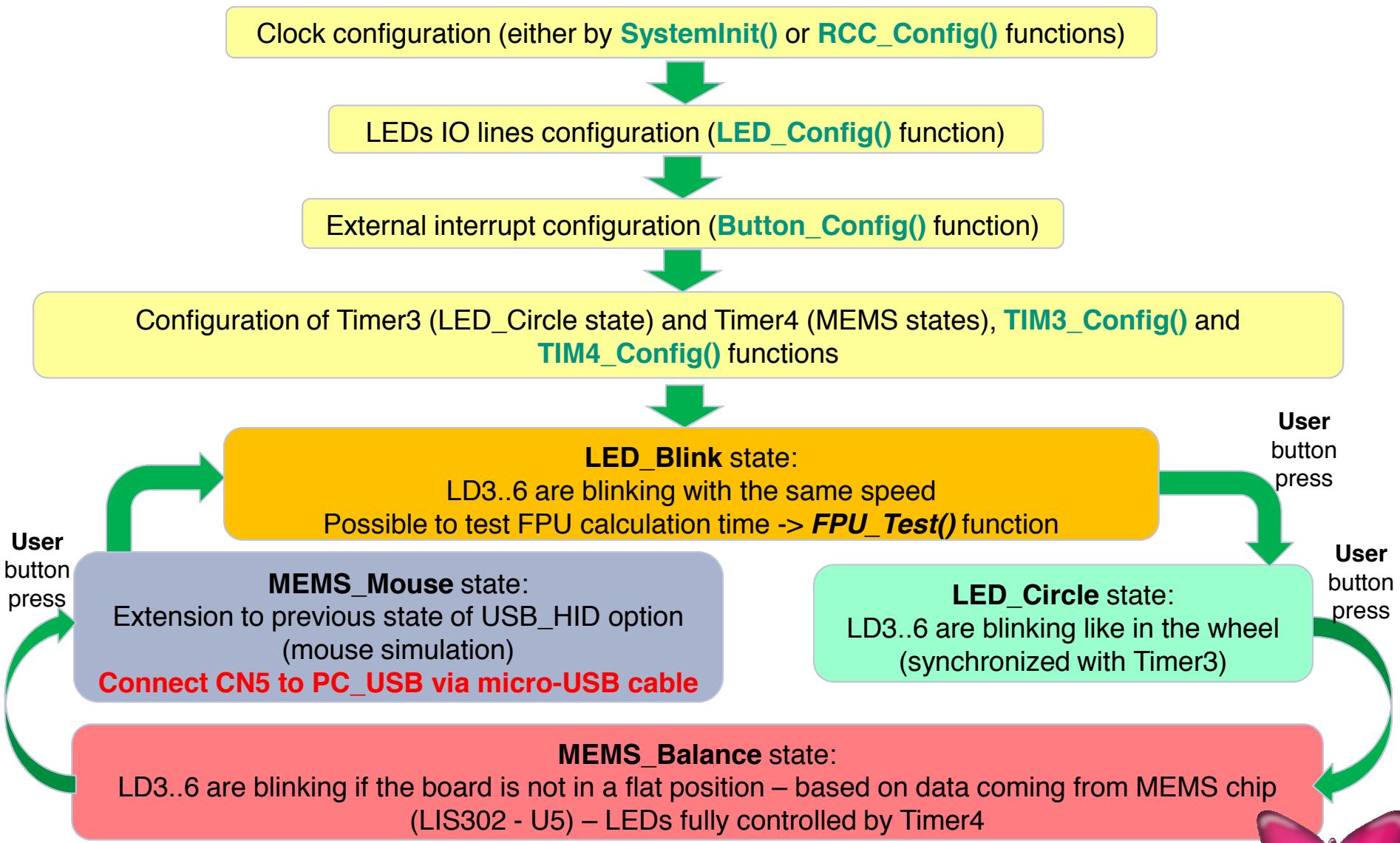
Basic configuration exercise 1/2

- Open the workspace and project in Keil uVision (simple double click on the file):
./2Do/Ex1/Project/Ex1-STM32F4x-Discovery demo.uvproj
- The source code is damaged in few places (only few files from ***_Application*** folder).
- Modification should be done only in:
 - ***main.c*** -> main procedures
 - ***stm32f4xx_it.c*** -> interrupt procedures,
 - ***stm32f4xx_conf.h*** -> selection of correct modules from the library
- Places to be modified are marked “?” symbols and are combined into 5 mini-tasks fully described in coming sections.
- Manual to the library is located in ./2Do folder:
stm32f4xx_dsp_stdperiph_lib_um.chm

The task is to detect and eliminate all the issues in order to make program run on STM32F4-Discovery board in line with the algorithm from the next slide.



Basic configuration exercise 2/2



GPIO configuration - theory

- After the **reset** all pins are in **input floating** mode
- Pins are grouped into 16bit ports (GPIOA, GPIOB, ... GPIOI)
- Most of the pins tolerates 5V as input signal
- GPIO ports are configured by several registers [names follow reference manual] which are updated by **GPIO_Init()** function automatically with the values from the **GPIO_InitTypeDef** structure:
 - **GPIO_Pin** -> GPIO_Pin_0 15, GPIO_Pin_All, GPIO_Pin_None
 - **GPIO_Mode**:
 - **GPIO_Mode_AN** //analog mode
 - **GPIO_Mode_IN** //input mode
 - **GPIO_Mode_OUT** //output mode
 - **GPIO_Mode_AF** //alternate function mode
 - **GPIO_OType**:
 - **GPIO_OType_PP**
 - **GPIO_OType_OD**
 - **GPIO_Speed**:
 - **GPIO_Speed_2MHz** //lowest EMI -> softer edges
 - **GPIO_Speed_25MHz**
 - **GPIO_Speed_50MHz**
 - **GPIO_Speed_100MHz** //highest EMI -> sharper edges
 - **GPIO_PuPd**:
 - **GPIO_PuPd_NOPULL**
 - **GPIO_PuPd_UP**
 - **GPIO_PuPd_DOWN**

GPIO configuration - task

Uncomment definition of **GPIO_task** in **main.h** file

Correct **LED_Config()** function (**main.c** file) in order to configure lines 12..15 from port GPIOD:

- In GPIO state (used in LED_Blink and LED_Circle states):
As general purpose output pins in push-pull configuration with 25MHz speed, without pull-up

- In other states (used in MEMS_Balance and MEMS_Mouse states):
As IO lines connected to Timer4 (as its outputs) with 25MHz speed (alternate function configuration)

Do not forget about the connection of the clock of used peripherals BEFORE the configuration



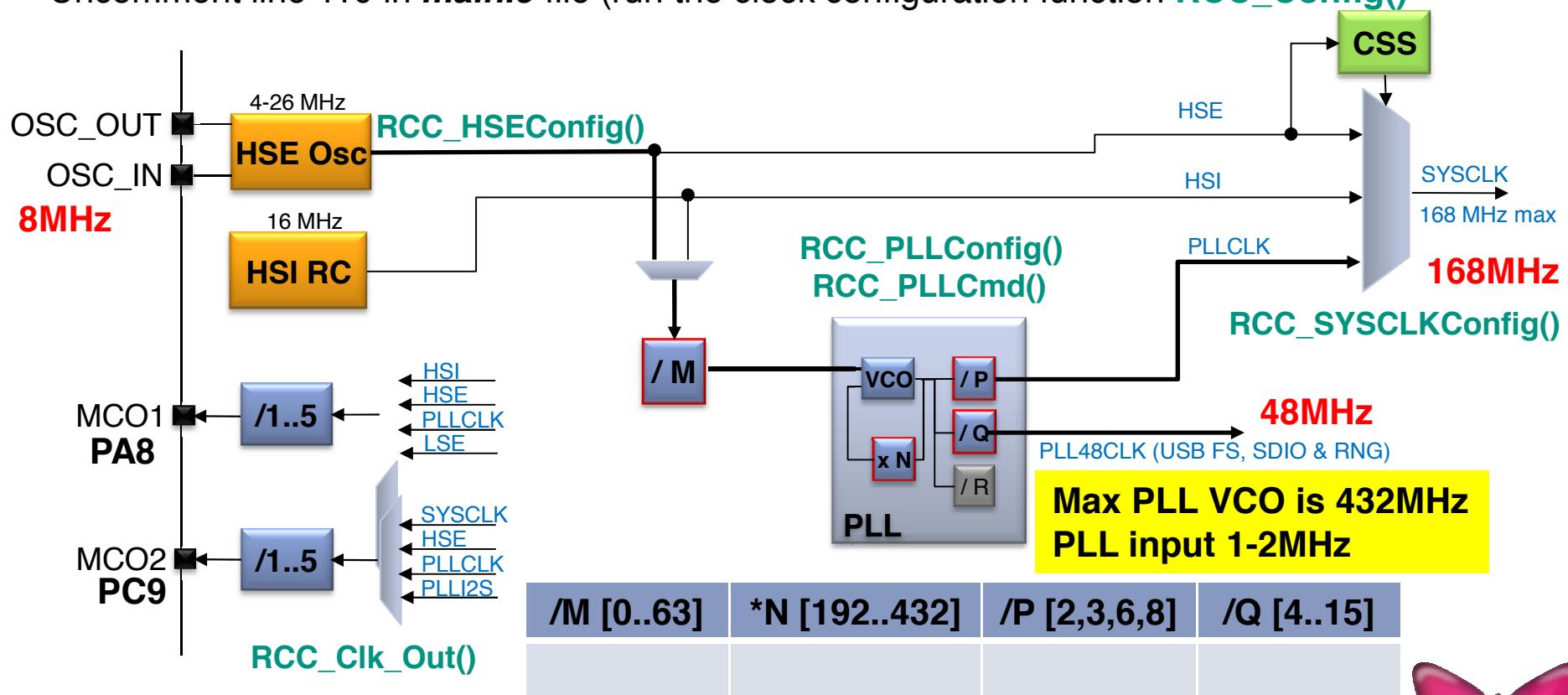
Clock configuration - theory

- After the reset system clock is set to HSI = 16MHz
- After the reset all peripherals (GPIOs as well) have clock disconnected
- There is Clock Security System for HSE monitoring. In case of problems with HSE - an automatic switch to HSI (reset state) occurs
- It is possible to send main clock to output pins (MCO1 or MCO2) -> up to 100MHz
- It is not possible to clock core and main peripherals by low speed oscillators (LSI and LSE).

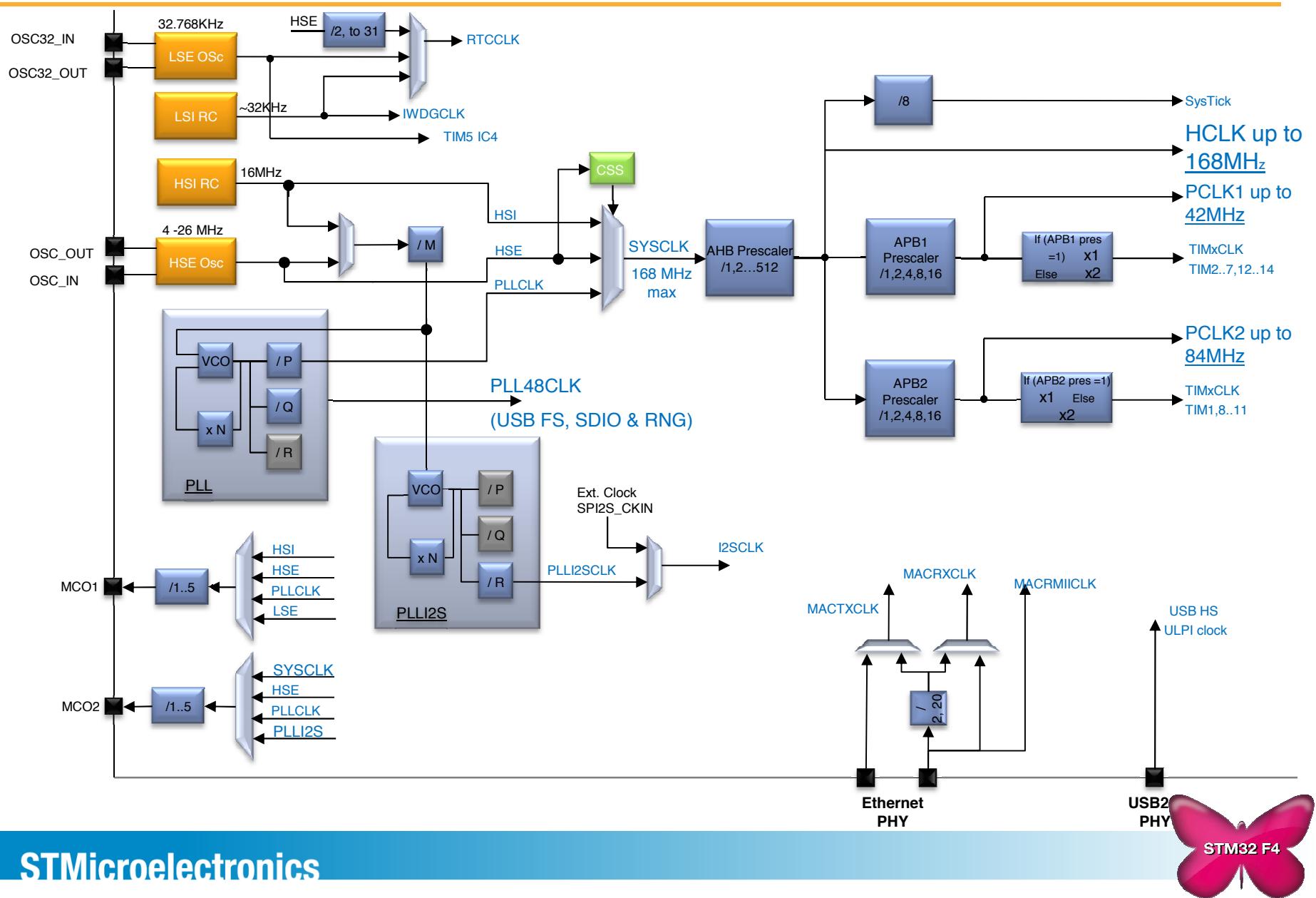
- When using STM32 standard firmware library there is an automatic clock configuration performed before main code, which switch clock to its maximum frequency (168MHz) based on HSE source. It is done by **SystemInit()** function called from startup file before the **main()**.

Clock configuration - task

- Uncomment definition of **CLK_task** in **main.h** file
- Calculate correct values of parameters **M,N,P,Q** and insert them in **RCC_Config()** function (**main.c** file). **Target frequencies** are put in **RED** on below diagram.
- Comment lines 178,179 (; style) in **startup_stm32f4xx.s** file – turn off initial clock configuration done by **SystemInit()** function
- Uncomment line 110 in **main.c** file (run the clock configuration function **RCC_Config()**)



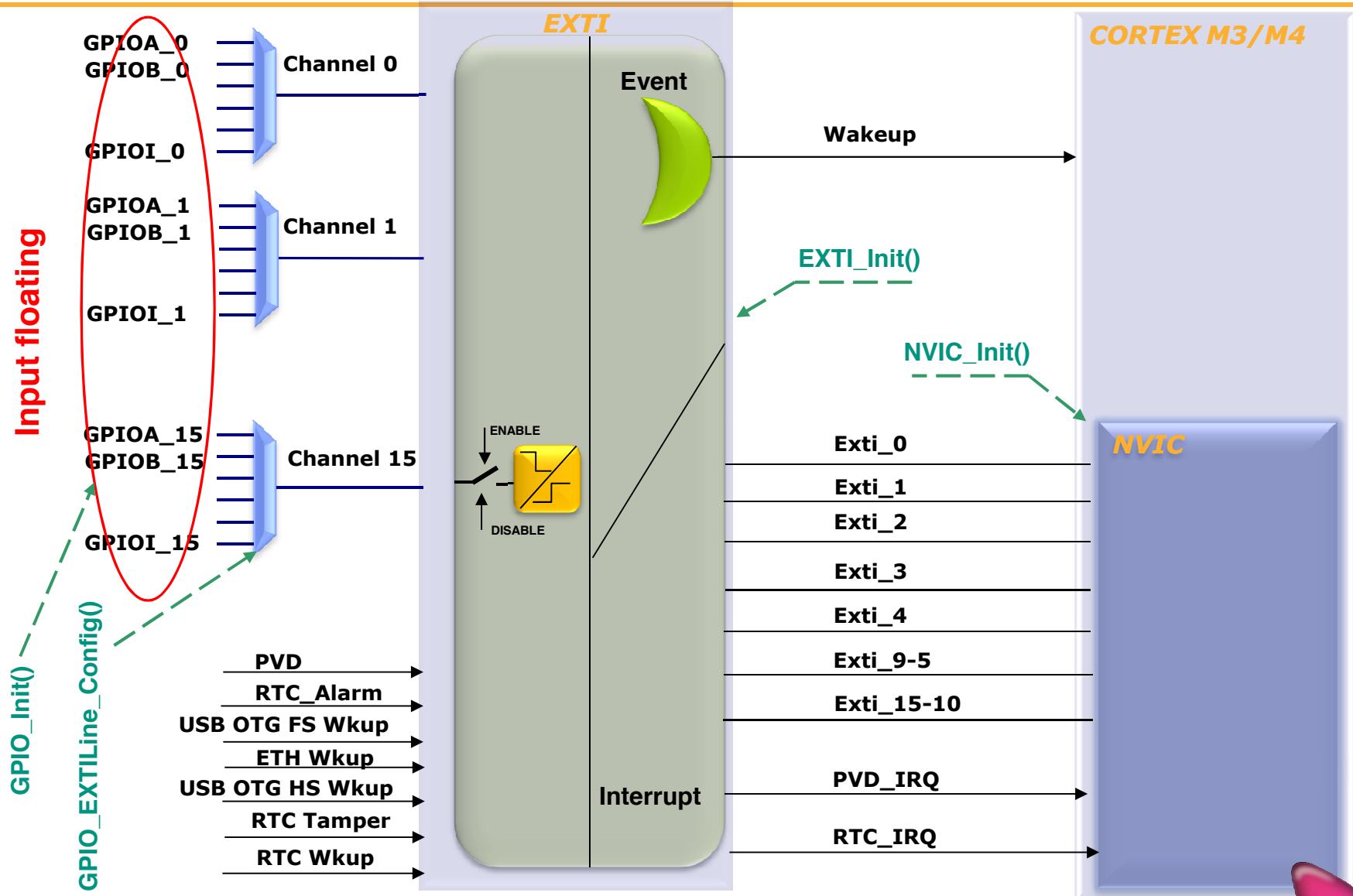
Clock Scheme – complete view



Interrupts - theory

- After the **reset** all peripheral **interrupts** are **disabled**, vector table is located at the beginning of the Flash memory
- Interrupt should be:
 - enabled at peripheral -> exact source of the interrupt
 - configured in NVIC (interrupt controller) -> priorities, location in memory
 - programmed in **stm32f4xx_it.c** -> body of its procedure
- In addition external interrupt requires:
 - Configuration of dedicated IO pin as **input** (GPIO module)
 - Specify if it will be **event or interrupt** mode (EXTI module)
 - Select **proper port to source interrupt** at the selected channel (i.e Channel 1 can be sourced by Pin 1 from any port (SYSCFG module))
 - Enable external interrupt channel (EXTI module)
 - Select **sensitivity** of the channel (raising or falling edge) (EXTI module)

EXTI module: from pin to NVIC



EXTI & NVIC configuration

- `GPIO_InitTypeDef` -> select input pin and configure it in input mode
- `GPIO_EXTILineConfig` -> configure input multiplexers
- `EXTI_InitTypeDef`:
 - `EXTI_Line` -> EXTI_Line0 15
 - `EXTI_Mode`:
 - EXTI_Mode_Event (for wakeup the core without interrupt generation)
 - EXTI_Mode_Interrupt
 - `EXTI_Trigger`:
 - EXTI_Trigger_Rising
 - EXTI_Trigger_Falling
 - `EXTI_LineCmd`:
 - ENABLE (turn on the channel)
 - DISABLE
- `NVIC_InitTypeDef`:
 - `NVIC_IRQChannel`: PPP_IRQHandler *)
 - `NVIC_IRQChannelPreemptionPriority`: 0..15 (lower number, higher priority)
 - `NVIC_IRQChannelSubPriority`: 0..15
 - `NVIC_IRQChannelCmd` -> ENABLE/DISABLE

*) PPP – name of interrupt vector defined in `stm32f4xx.h` (or described in library manual)



Interrupts - task

Uncomment definition of **BUTTON_task** in **main.h** file

Correct **Button_Config()** function (**main.c** file) in order to make User button working:

- Configure GPIOA, pin0 as input (GPIO module)
- Configure pin as working with the port (no other peripheral) (SYSCFG module)
- Configure its mode (interrupt) and sensitivity (rising edge) (EXTI module)
- Configure interrupt vector and its priorities (NVIC module)

Correct interrupt vector function - **EXTI0_IRQHandler()** in **stm32f4xx_it.c** file

- Clear the interrupt flag:



Timer configuration - procedure

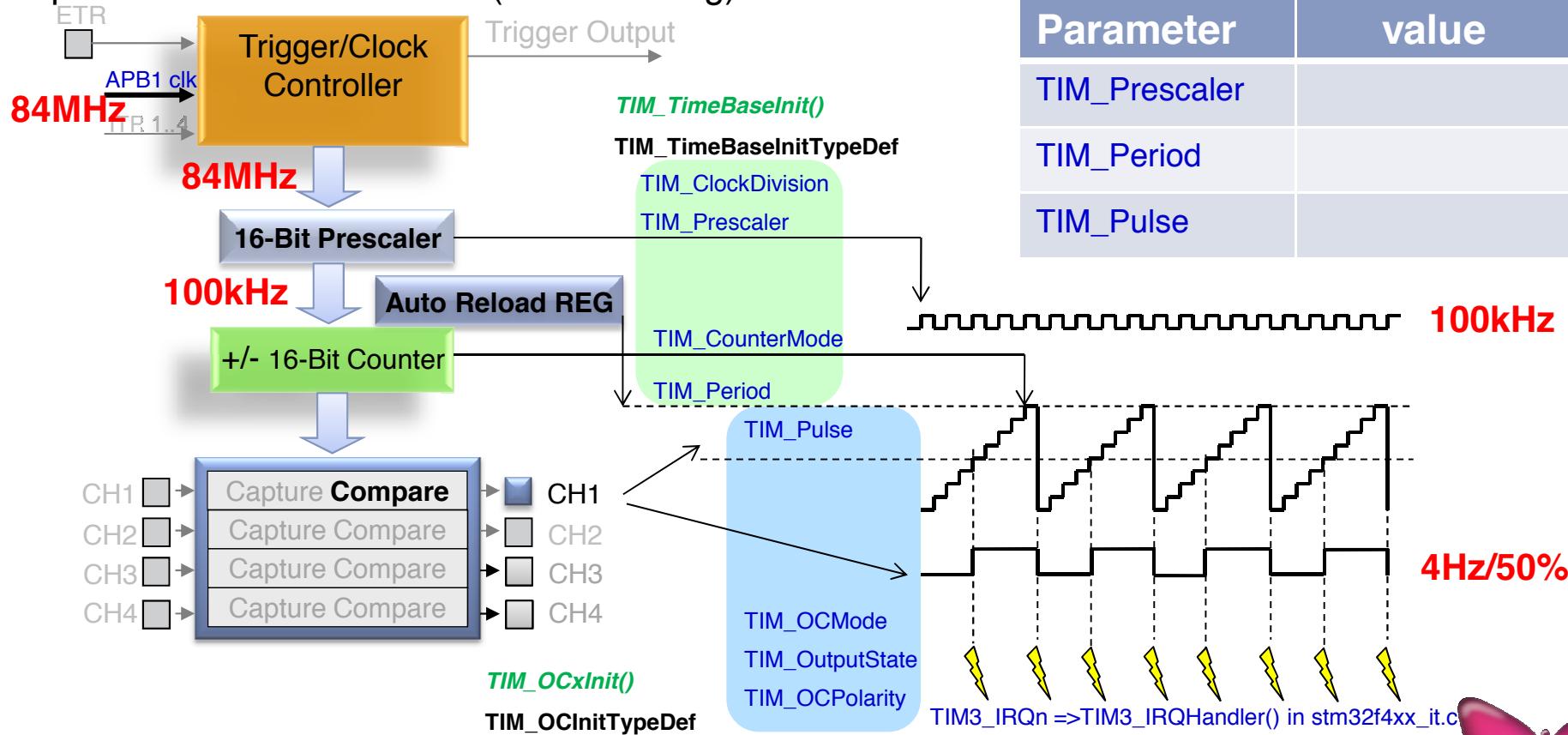
- Turn on Timer clock at RCC module (APBx bus) using function **RCC_APBxPeriphClockCmd()**
- Configure timer base module using **TIM_TimeBaseInitTypeDef** structure for selected timer and then function **TIM_TimeBaseInit()**
- Configure **TIM_OCInitTypeDef** structure and then function **TIM_OCxInit()** for selected channels of the timer.
- Initiate preload autoreload register using **TIM_OCxPreloadConfig()** function and preload of capture compare registers using **TIM_ARRPreloadConfig()** function for each used channel
- Enable the timer using **TIM_Cmd()** function

Timer configuration - structures

- There are two structures to be filled in in order to configure selected channel to PWM generation:
 - Time Base -> **TIM_TimeBaseInitTypeDef**
 - **TIM_Period** – autoreload value
 - $f_{pwm} = TIM_counter_clk / (\text{Period} + 1)$
 - **TIM_Prescaler** - [0 -> $2^{16}-1$] – TIM3 input clock prescaler value
 - $\text{TIM_counter_clk} = \text{APB1_clk} / (\text{prescaler} + 1)$
 - **TIM_ClockDivision** – used for input digital filters, can be left 0
 - 0
 - **TIM_CounterMode** – TIM_CounterMode_ [Up/Down/CenterAligned1..3]
 - TIM_CounterMode_Up
 - Capture Compare section for channel x -> **TIM_OCInitTypeDef**
 - **TIM_OCMode** – different configurations for Output Compare mode
 - TIM_OCMode_PWM1
 - **TIM_OutputState** – input mode: capture enable, output mode: output enable
 - TIM_OutputState_Enable
 - **TIM_Pulse** – [0 -> $2^{16}-1$] – capture compare for channel x register value
 - $\text{Duty_cycle} = (\text{TIM_Pulse} / \text{TIM_Period}) * 100\%$
 - **TIM_OCPolarity** – output signal active high or low
 - TIM_OCPolarity_High

Timer3 – Output Compare mode - task

- Uncomment definition of **TIMER3_task** in **main.h** file
- Correct **TIM3_Config()** function in **main.c** file in order to configure Timer3 in output compare mode to generate update/overflow interrupts with 4Hz frequency and in the meantime capture/compare interrupt on channel 1. In Timer3 interrupt routine there is a control of LD3..6.
- Result: in LED_Circle state of main loop LD3..6 should blink with 2Hz frequency and with phase shift to the next one (circle flashing)



FPU - task

- inside the code there is a function **FPU_Test()** within LED_Blink part of the main loop of the code
- within the function there are two random floating point number generated and basic operations are performed.
- by usage of System Timer (SysTick) it is possible to check how many system clock cycles each operation on two floating point argument is using:

- time_add** – addition
- time_sub** – subtraction
- time_mul** – multiplication
- time_div** – division

- using Keil uVision configuration it is possible to turn on and off the hardware support for floating point operations

No FPU	FPU

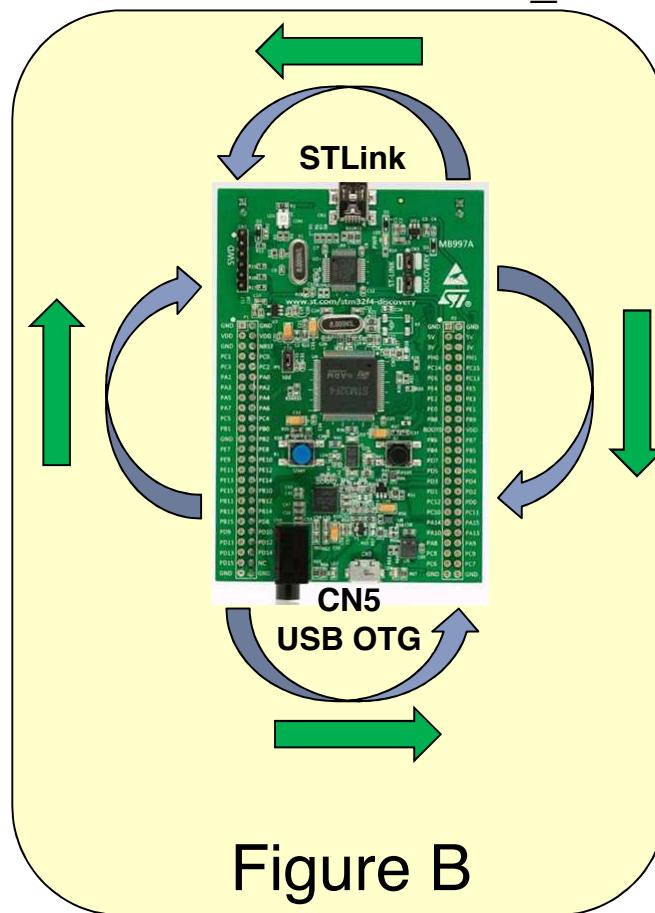
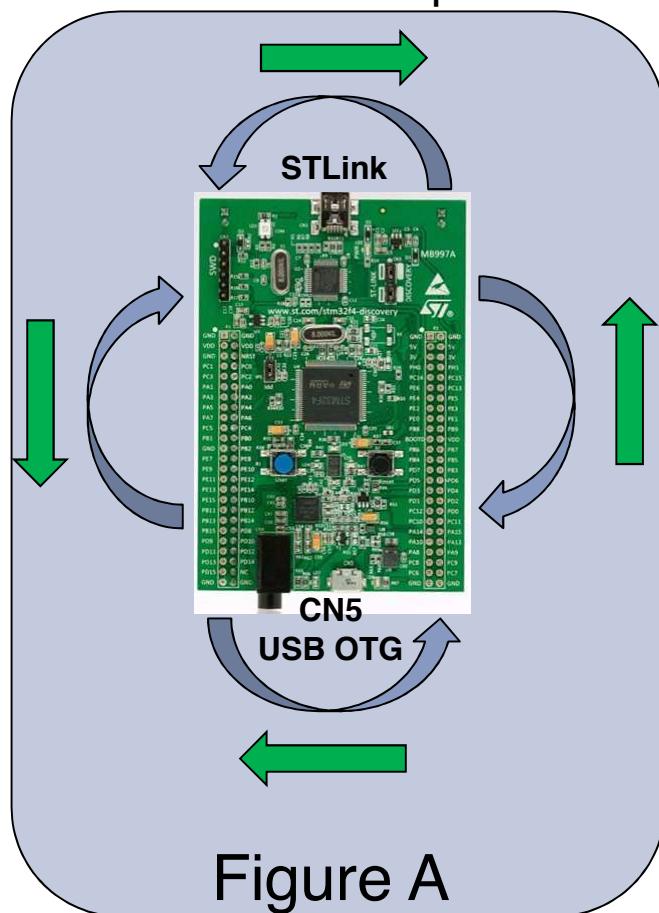


TASK: please check how many clock cycles uses each operation with and without hardware floating point support from the core (usage of FPU module)



Fine tuning of the application - task

- The MEMS_Mouse state is working in so called “reverse orientation” like in the plane, like on figure A.
- The task would be to change way of operation to work like on the figure B. The answer to this question is located in the file ***stm32f4xx_it.c***.



PC cursor movement direction
 Board movement direction