STM32 Ecosystem workshop

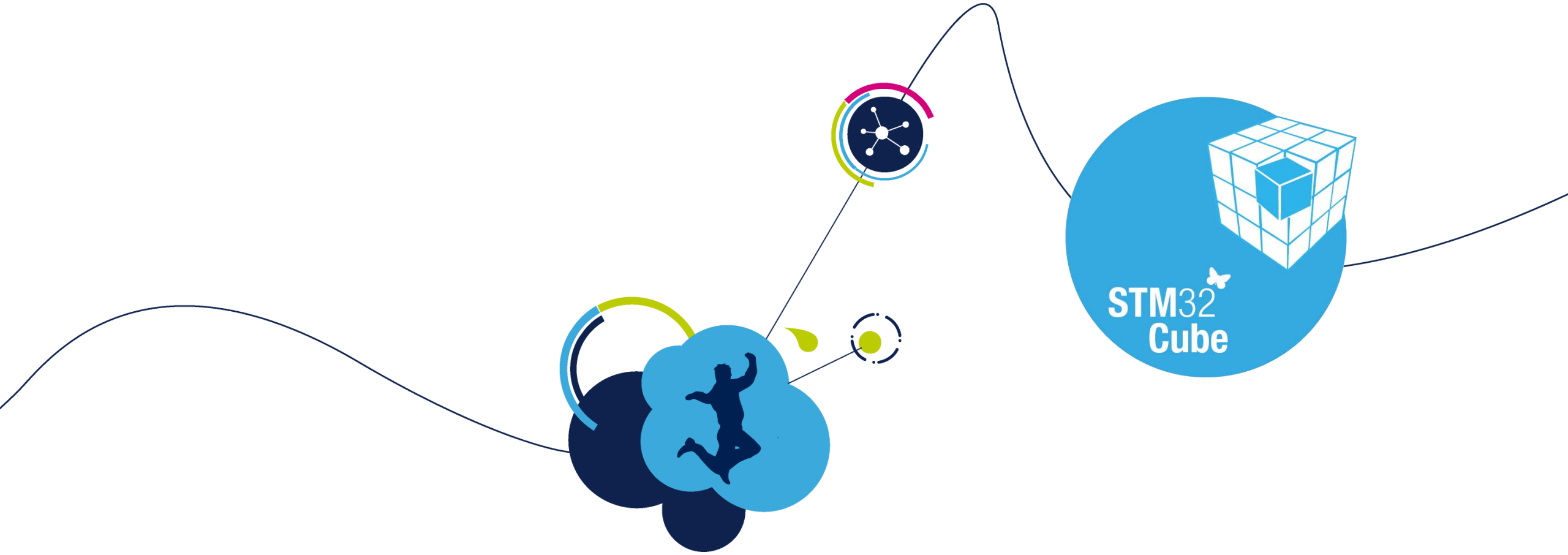
T.O.M.A.S Team

life.augmented

- Before adding a new code let’s go with some theory explaining the Cube Library structure and what is generated by STM32CubeMX

Goal of this part

- Understand the structure of the code generated by STM32CubeMX
  - Know the role of the library files
  - Know the role of the functions executed before main
  - Understand interrupt handling process



STM32CubeMX generated code

life.augmented

STM32CubeMX generated code

code structure, main operations after the reset

STM32CubeMX

```
SystemInit();
main()
{
      HAL_Init()
      {
       HAL_MspInit();
      }

      SystemClockConfig();

      MX_PPP_Init()
      {
        HAL_PPP_Init()
        {
         HAL_PPP_MspInit();
        }
      }
```

User

```
      HAL_PPP_Action();
      while(1);
}
```

Located in: `system_stm32l4xx.c`
Called from: `startup_stm32l4xx.s` (reset vector before main())

It performs the following operations:

- Enables Floating Point Unit (**FPU**) inside the core
- Resets the **Clock Configuration** to the default reset state
- Disables all **Interrupts**
- Configures **Vector Table** location and its offset

It does NOT configure the clock system (as it was done in Standard Peripherals Library (SPL))

STM32CubeMX generated code

code structure, main operations after the reset

STM32CubeMX

```
SystemInit();
main()
{
      HAL_Init()
      {
       HAL_MspInit();
      }

      SystemClockConfig();

      MX_PPP_Init()
      {
        HAL_PPP_Init()
        {
         HAL_PPP_MspInit();
        }
      }
```

User

```
      HAL_PPP_Action();
      while(1);
}
```

Located in: `stm32l4xx_hal.c`
Called from: `main.c`

It performs the following operations:

- Configures **FLASH accelerator**
- Configures **NVIC priority** (group and sub-priorities split)
- Configures **SysTick** for 1ms tick (based on HSI clock)
- Initializes **Low-level hardware** selected in STM32CubeMX (call to HAL_MspInit() function)

STM32CubeMX generated code

code structure, main operations after the reset

STM32CubeMX

```
SystemInit();
main()
{
      HAL_Init()
      {
       HAL_MspInit();
      }

      SystemClockConfig();

      MX_PPP_Init()
      {
        HAL_PPP_Init()
        {
         HAL_PPP_MspInit();
        }
      }
```

User

```
      HAL_PPP_Action();
      while(1);
}
```

Located in: `stm32l4xx_hal_msp.c`
Called from: `stm32l4xx_hal.c`

It performs the following operations:

- Configures **Interrupts Priorities**
  (for each peripheral selected in STM32CubeMX)

STM32CubeMX generated code

code structure, main operations after the reset

STM32CubeMX

```
SystemInit();
main()
{
      HAL_Init()
      {
       HAL_MspInit();
      }

      SystemClockConfig();

      MX_PPP_Init()
      {
        HAL_PPP_Init()
        {
         HAL_PPP_MspInit();
        }
      }
```

User

```
      HAL_PPP_Action();
      while(1);
}
```

Located in: `main.c`
Called from: `main.c`

It performs the following operations:

- Configures **System & Buses clocks** (based on STM32CubeMX clock settings)

STM32CubeMX generated code

code structure, main operations after the reset

STM32CubeMX

```
SystemInit();
main()
{
      HAL_Init()
      {
       HAL_MspInit();
      }

      SystemClockConfig();

      MX_PPP_Init()
      {
        HAL_PPP_Init()
        {
         HAL_PPP_MspInit();
        }
      }
```

User

```
      HAL_PPP_Action();
      while(1);
}
```

Located in: `main.c`
Called from: `main.c`

It performs the following operations:

- Initializes **PPP peripheral** based on STM32CubeMX configuration
  - Uses structures and dedicated initialization functions type HAL_PPP_Init()
  - Within those functions this is possible to tune a peripheral configuration

STM32CubeMX generated code

code structure, main operations after the reset

STM32CubeMX

```
SystemInit();
main()
{
      HAL_Init()
      {
       HAL_MspInit();
      }

      SystemClockConfig();

      MX_PPP_Init()
      {
        HAL_PPP_Init()
        {
         HAL_PPP_MspInit();
        }
      }
```

User

```
      HAL_PPP_Action();
      while(1);
}
```

Located in: `stm32l4xx_hal_PPP.c`
Called from: `main.c`

It performs the following operations:

- Initializes the **PPP peripherial mode (**according to parameters specified in the PPP_InitTypeDef structure) **and** the associated **handle**

STM32CubeMX generated code

code structure, main operations after the reset

STM32CubeMX

```
SystemInit();
main()
{
      HAL_Init()
      {
       HAL_MspInit();
      }

      SystemClockConfig();

      MX_PPP_Init()
      {
        HAL_PPP_Init()
        {
         HAL_PPP_MspInit();
        }
      }
```

User

```
      HAL_PPP_Action();
      while(1);
}
```

Located in: `stm32l4xx_hal_msp.c`
Called from: `stm32l4xx_hal_PPP.c`

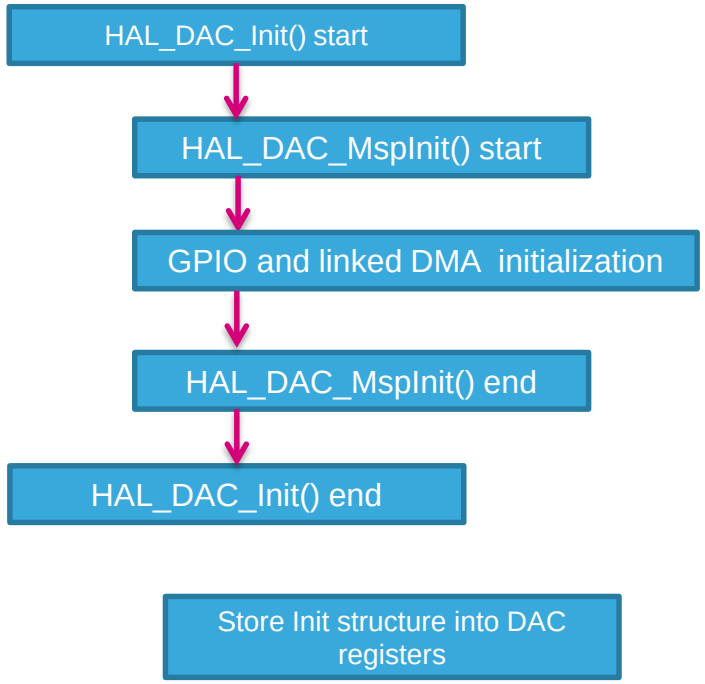
It performs the following operations:

- Configures **Clocks** related to PPP peripheral
- Configures **GPIO lines** assigned to PPP peripheral
- Configures **DMA channel** assigned to PPP peripheral (except source and destination address and number of the data to be transferred – this is done by HAL_PPP_Action() function)
- Configures **Interrupts** selected for PPP peripheral

STM32CubeMX generated code

peripherals initialization (DAC example)

- HAL_DAC_Init() function details



All HAL_DAC_Init() functions are calling the same function HAL_DAC_MspInit()

Handler instance parameter is used to determine which DAC needs to be initialized

Initialize GPIO lines selected in STM32CubeMX

Initialize DMA channel selected in STM32CubeMX

```
void HAL_DAC_MspInit(DAC_HandleTypeDef* hdac)
{
  GPIO_InitTypeDef GPIO_InitStruct;
  if(hdac->Instance==DAC)
  {
  /* USER CODE BEGIN DAC_MspInit 0 */

  /* USER CODE END DAC_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_DAC_CLK_ENABLE();

    /**DAC GPIO Configuration
    PA4     ------> DAC_OUT1
    */
    GPIO_InitStruct.Pin = GPIO_PIN_4;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    /* Peripheral DMA init*/

    hdma_dac_ch1.Instance = DMA1_Channel2;
    hdma_dac_ch1.Init.Request = DMA_REQUEST_9;
    hdma_dac_ch1.Init.Direction = DMA_MEMORY_TO_PERIPH;
    hdma_dac_ch1.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_dac_ch1.Init.MemInc = DMA_MINC_ENABLE;
    hdma_dac_ch1.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
    hdma_dac_ch1.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
    hdma_dac_ch1.Init.Mode = DMA_CIRCULAR;
    hdma_dac_ch1.Init.Priority = DMA_PRIORITY_LOW;
    if (HAL_DMA_Init(&hdma_dac_ch1) != HAL_OK)
    {
      Error_Handler();
    }
    __HAL_LINKDMA(hdac,DMA_Handle1,hdma_dac_ch1);

  /* USER CODE BEGIN DAC_MspInit 1 */

  /* USER CODE END DAC_MspInit 1 */
  }
}
```

STM32CubeMX generated code

code structure, main operations after the reset

STM32CubeMX

```
SystemInit();
main()
{
      HAL_Init()
      {
       HAL_MspInit();
      }

      SystemClockConfig();

      MX_PPP_Init()
      {
        HAL_PPP_Init()
        {
         HAL_PPP_MspInit();
        }
      }
```

User

```
      HAL_PPP_Action();
      while(1);
}
```

Located in: `main.c`
Called from: `main.c`

It performs the following operations:

- PPP **peripherals control** (i.e. Start, Stop, Calibration)
- **DMA configuration** if needed (buffer location, source and destination addresses, number of data to be transferred)

STM32CubeMX generated code

code structure, main operations after the reset - summary


```
SystemInit();
main()
{
        HAL_Init()
        {
         HAL_MspInit();
        }

        SystemClockConfig();

        MX_PPP_Init()
        {
          HAL_PPP_Init()
          {
           HAL_PPP_MspInit();
          }
        }

        HAL_PPP_Action();
        while(1);
}
```

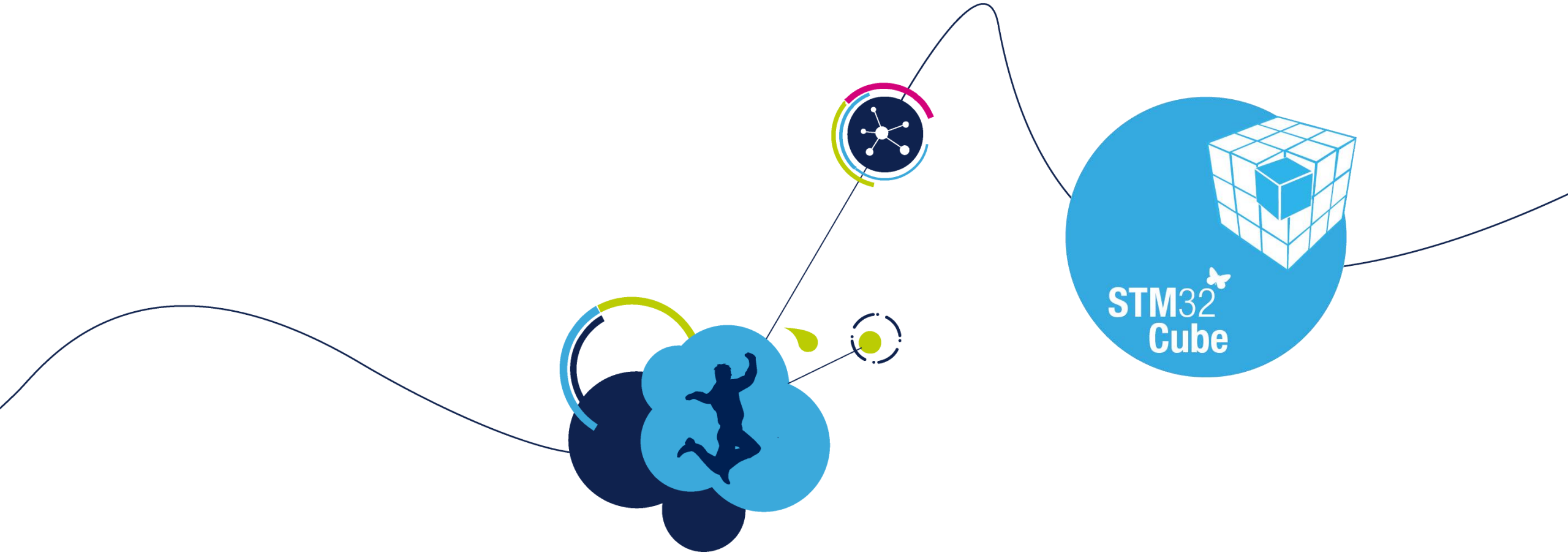
Operation	Function • called from • source location	Origin
• Enable Floating Point Unit (FPU) inside the core • Reset the clock configuration to the default reset state • Disable all interrupts • Configure vector table location and its offset	**SystemInit()** • startup_stm32xxxx.s • system_stm32xxxx.c	STM32CubeMX
• Configure FLASH accelerator • Configure NVIC priority (group and sub-priorities split) • Configure SysTick for 1ms tick (based on HSI clock) • Initialization of low level hardware selected in STM32CubeMX (HAL_MspInit() function)	**HAL_Init()** • main.c • stm32xxxx_hal.c	STM32CubeMX
• Configure Interrupts priorities (for each peripheral selected in STM32CubeMX)	**HAL_MspInit()** • stm32xxxx_hal_msp.c • stm32xxxx_hal_msp.c	STM32CubeMX
• Configure System & Buses clocks based on STM32CubeMX clock settings	**SystemClockConfig()** • main.c • main.c	STM32CubeMX
• Initialize PPP peripheral based on STM32CubeMX configuration:.PPP related peripheral configuration (GPIO lines, DMA channel) in function HAL_PPP_MspInit()	**MX_PPP_Init()** • main.c • main.c or PPP.c	STM32CubeMX
• Connect Clock to the peripheral • Configure GPIO lines • Configure DMA channels (based on STM32CubeMX settings)	**HAL_PPP_MspInit()** • stm32xxxx_hal_msp.c • stm32xxxx_hal_msp.c	STM32CubeMX
• Activate PPP peripherals control (i.e. Start, Stop, Calibration), • Configure DMA if needed (buffer location, source and destination addresses, number of data to be transferred)	**HAL_PPP_Action()** • main.c • user_code.c	User

STM32CubeMX generated code

code structure - interrupts


```
PPP_IRQHandler()
{
        HAL_PPP_Callback();
}
```

operation	Function • called from • source location	Origin
• Check all possible flags and interrupt triggers • Clear all interrupts flags • Call proper callback functions assigned to the particular interrupt source/trigger	**`PPP_IRQHandler()`** • stm32xxxx_it.c • stm32xxxx_hal_ppp.c	STM32CubeMX
• Perform an action related to the specific interrupt.	**`HAL_PPP_Callback()`** • user_code.c • user_code.c	User



HAL libraries

basic information

life.augmented

HAL library
function naming convention

HAL_StatusTypeDef **HAL_PPP(Ex)_Operation_mode()**

Return value:
- HAL_OK
- HAL_ERROR
- HAL_BUSY
- HAL_TIMEOUT
- **void** for simple peripherals (i.e. GPIO)

HAL prefix indicating type of the library (in contrary to Low Layer (LL)

PPP – peripherheral name. If 'Ex' suffix is added it means that this particular function is related to current MCU family line and cannot be directly migrated to other STM32 lines

Type of the operation on the peripheral, like 'Start', 'Stop'

Mode of the operation:
- polling – no suffix
- Interrupt – suffix IT
- DMA – suffix DMA

See next slides for further details

Hint: to find proper function for the peripheral, please type **`HAL_PPP_`** and press **Ctrl+Space**

HAL library
actions on peripherals

- Most of the configuration procedures are prepared and generated by STM32CubeMX based on user configuration
- After this process, within main() function, user should prepare series of the functions which would activate the selected peripherals in required mode:
  - Polling mode (function type `HAL_PPP_Action()`)
  - Interrupt mode (function type `HAL_PPP_Action_IT()`)
  - DMA mode (function type `HAL_PPP_Action_DMA()`)

Hint: to find proper function for the peripheral, please type **`HAL_PPP_`** and press **Ctrl+Space**

What have we learnt?

- ✓ Understand the structure of the code generated by STM32CubeMX
  - ✓ Know the role of the library files
  - ✓ Know the role of the functions executed before main
  - ✓ Understand interrupt handling process

Further reading

More information can be found in the following documents:

- **UM1860** - Getting started with STM32CubeL4 for STM32L4 Series, available on the web:

  http://www.st.com/resource/en/user_manual/dm00157440.pdf

- **UM1884** - Description of STM32L4 HAL and Low-layer drivers, available on the web:

  http://www.st.com/resource/en/user_manual/dm00173145.pdf

- Doxygen based html manual: **STM32L486xx_User_Manual.chm**, available within STM32L4xx Cube library in the path:

  \STM32Cube_FW_L4_V1.5.0\Drivers\STM32L4xx_HAL_Driver\

Enjoy!

www.st.com/mcu

life.augmented