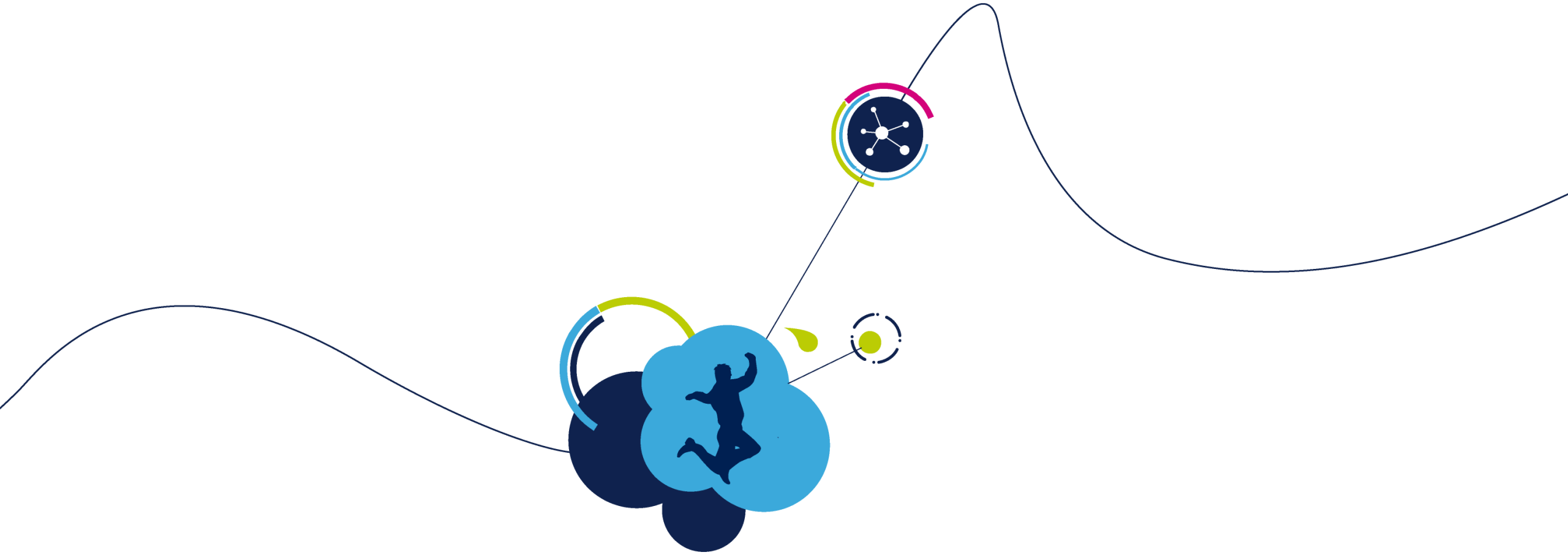
STM32 Ecosystem workshop

T.O.M.A.S Team

life.augmented

We will continue a bit more with software activities.

- Let’s try to rewrite our L4_DAC_ADC application using Low Layer libraries.
- In this step we will create an empty STM32CubeMX template and then, we will try to write complete application (except clock configuration) using only Low Layer library. Then we will compare the code size
- We have two ways to complete this task:
  - Usage of unitary init functions requiring good knowledge of the peripherals (following reference manual configuration steps)
  - Usage init functions (similar method to Standard Peripherals Library)
- Let’s try second option, then we can compare the code size of the projects
- What would be the difference?

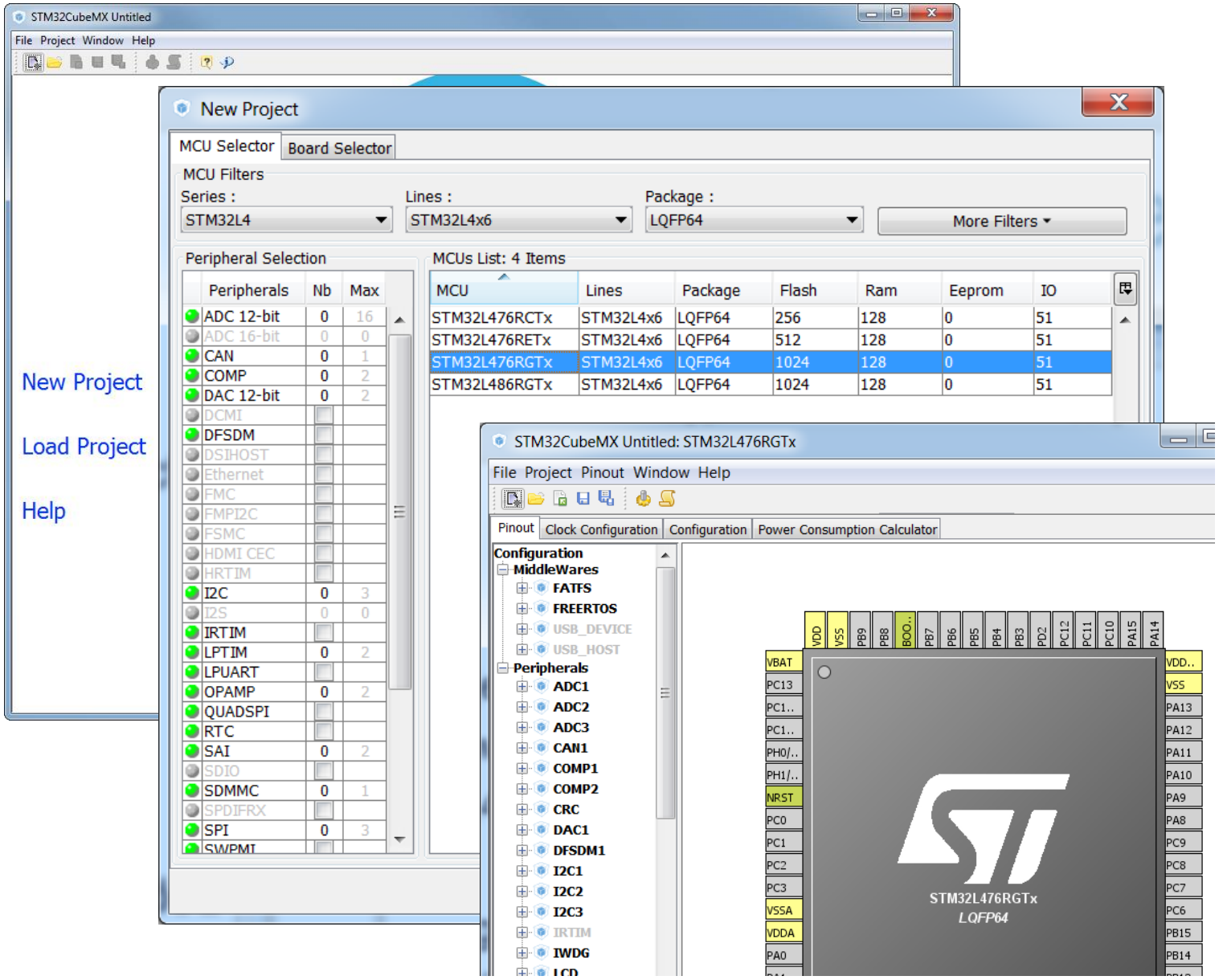


Creating the L4_DAC_ADC project with full usage of LL library

life.augmented

New project creation

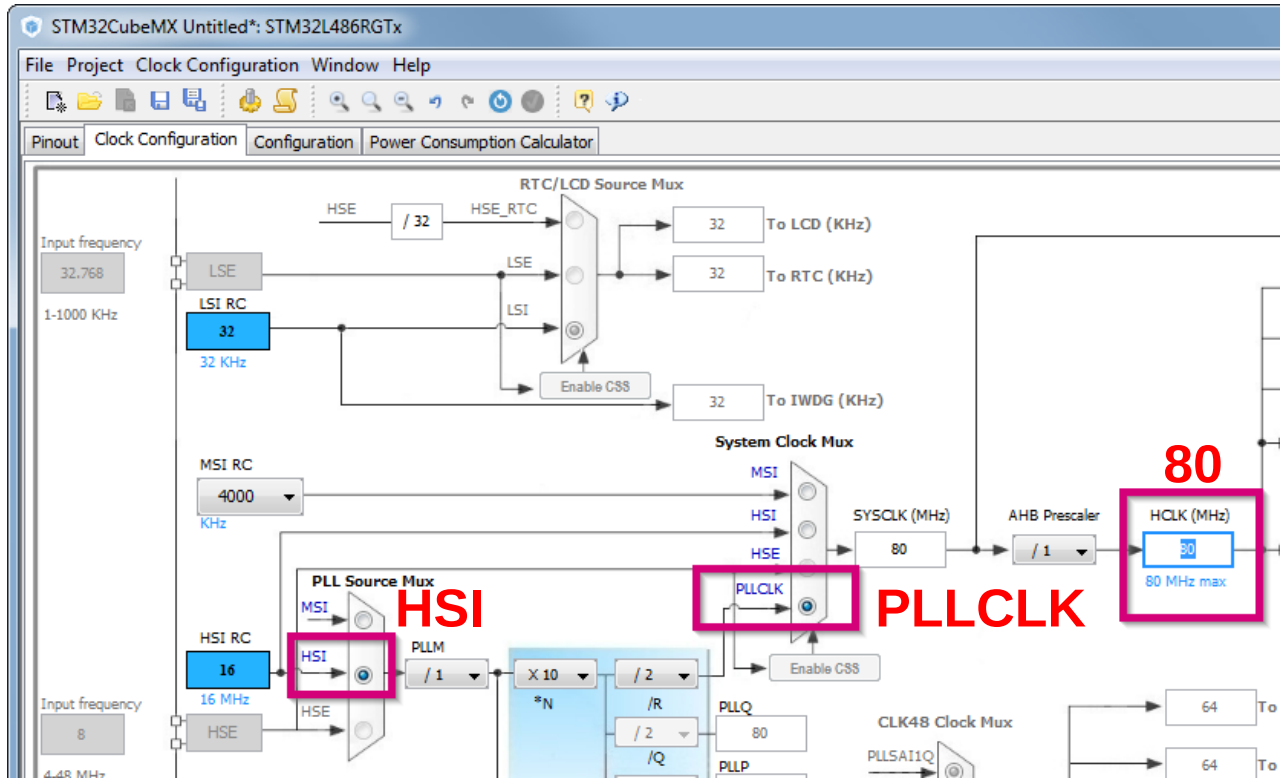
- Open STM32CubeMX
- Select **New Project**
- *Menu → File → New project*
- Select STM32L4
- STM32L4x6
- LQFP64 package
- STM32L476RGTx
- Do not select any peripherals (will be added manually in software)

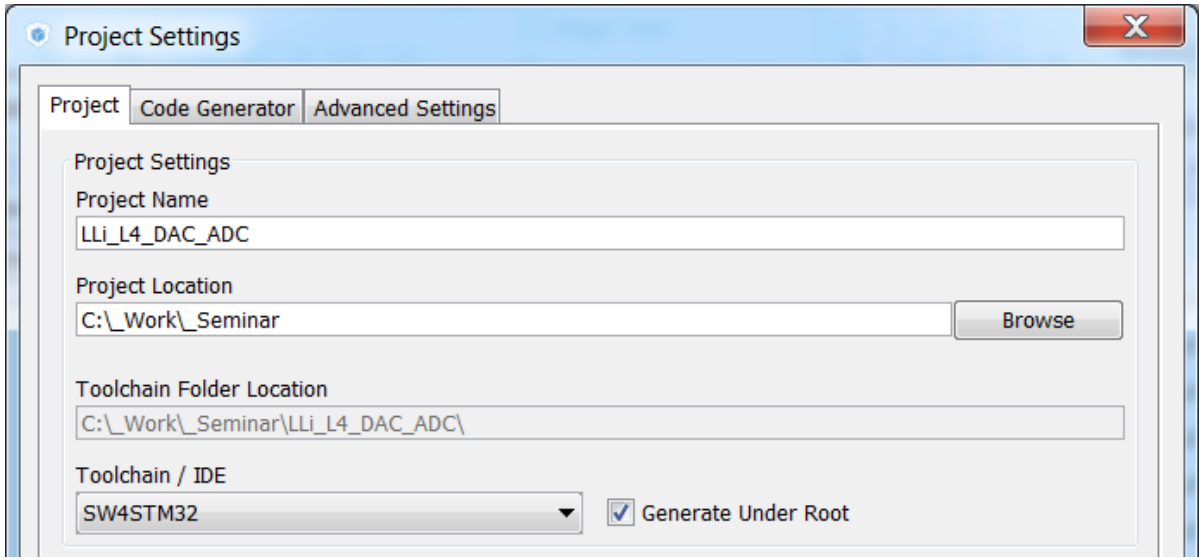


life.augmented

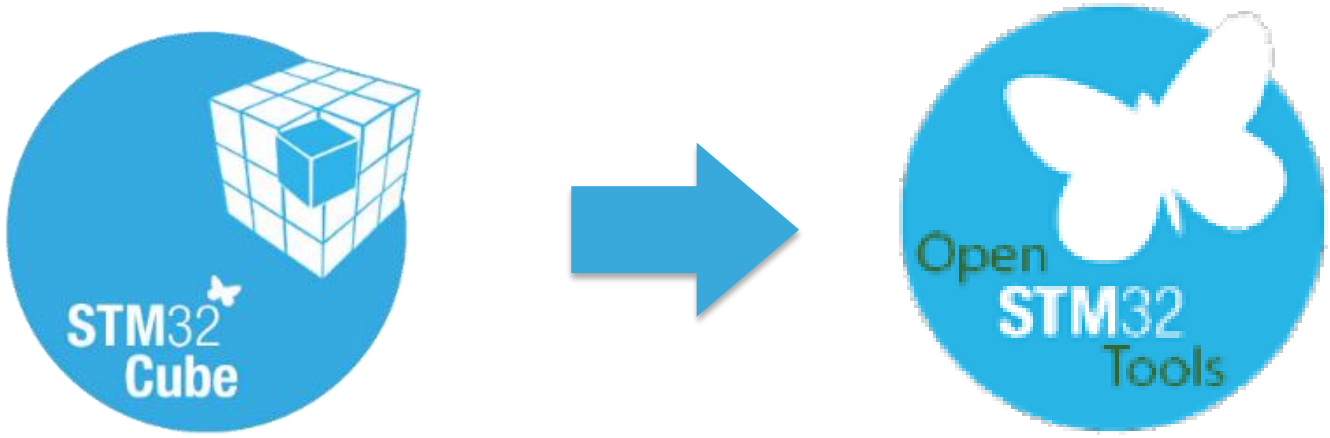
Template project generation

- Go to **Clock Configuration** Tab
- Set clocks to 80MHz (based on 16MHz HSI*PLL)
- Save project as **LL_L4_DAC_ADC_Init**
- Generate the C project for SW4STM32
- Import new project into the SW4STM32 workspace used in previous exercises



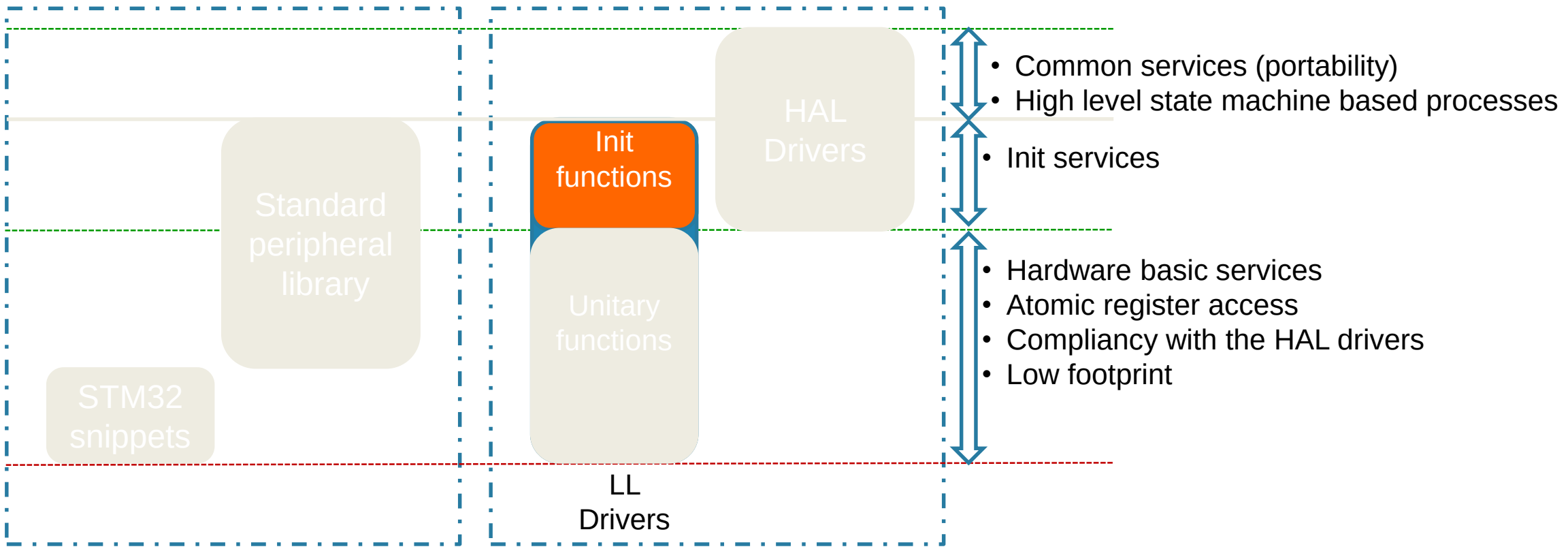


- Generate the code with added new features
- Open newly generated project in SW4STM32



LL_L4_DAC_ADC_Init

using init functions



Copy LL library files into the project

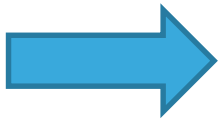
- From repository *CubeMX → Menu → Help → Updater Settings* get repository path (marked **.** below)
- From **.\STM32Cube_FW_L4_Vx.x.x\Drivers\STM32L4xx_HAL_Driver\src** copy:

```
stm32l4xx_ll_adc.c
stm32l4xx_ll_dac.c
stm32l4xx_ll_dma.c
stm32l4xx_ll_gpio.c
stm32l4xx_ll_rcc.c
stm32l4xx_ll_tim.c
```

$PROJ_DIR\Drivers\STM32L4xx_HAL_Driver\Src\

- From **.\STM32Cube_FW_L4_Vx.x.x\Drivers\STM32L4xx_HAL_Driver\inc** copy:

```
stm32l4xx_ll_adc.h
stm32l4xx_ll_bus.h
stm32l4xx_ll_dac.h
stm32l4xx_ll_dma.h
stm32l4xx_ll_gpio.h
stm32l4xx_ll_rcc.h
stm32l4xx_ll_tim.h
```



$PROJ_DIR\Drivers\STM32L4xx_HAL_Driver\Inc\

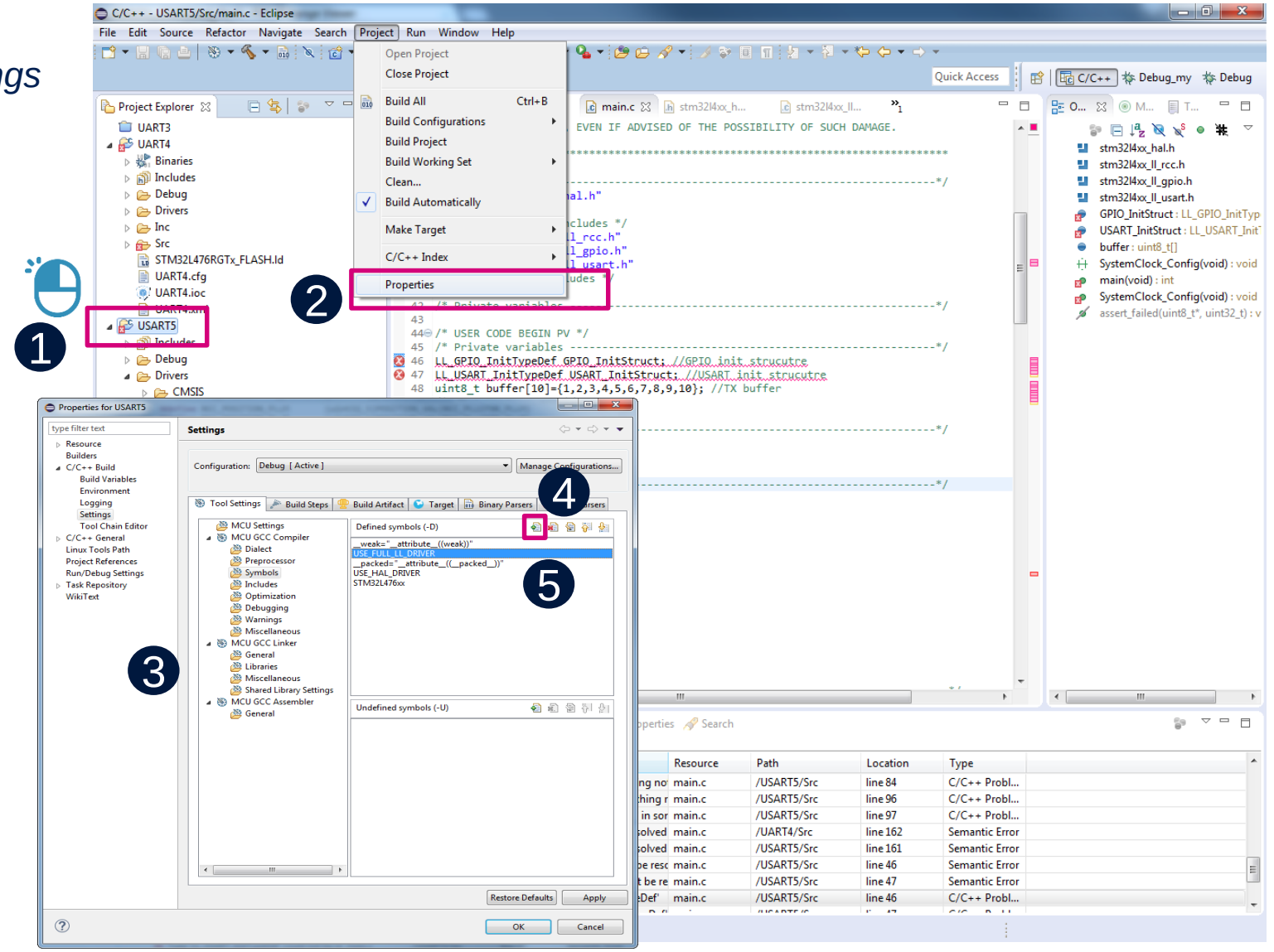
- Refresh (F5) the project source files – now, new files will become visible

Project configuration

enable usage of init functions in Low Layer Library

1. Select project
2. *Menu>Project>Properties>C/C++ Build>Settings*
3. MCU GCC Compiler>Symbols
4. Add …
5. Define **USE_FULL_LL_DRIVER** to allow use of LL library

It will allow us to use the Init…() functions like in Standard Peripherals Library (SPL)



life.augmented

Writing the code

LL_L4_DAC_ADC_Init project - tasks

Within **`main.c`** file perform the following actions

1. Include necessary low layer header files.
2. Declare initialization structures for used peripherals and buffers for DAC and ADC.
3. Initialize peripherals one by one (mind to connect the clock to the peripheral first).
4. Start the peripheral using low layer functions.

As a reference please use already copied header and source files for low layer part of the library.

Replacing HAL functions with init LL

1 - LL_L4_DAC_ADC_Init project - includes

- All used peripherals (**ppp**) need dedicated low layer header file **stm32l4xx_ll_ppp.h**
- We have to include them in **main.c** file in USER CODE section.
- There is a complete list of needed structures below. Please try to find and declare proper ones.

```
 /* USER CODE BEGIN Includes */
#include "stm32l4xx_ll_???.h"
…
…
…
…
…
…
/* USER CODE END Includes */
```

Replacing HAL functions with init LL

1 - LL_L4_DAC_ADC_Init project - includes

- All used peripherals (**ppp**) need dedicated low layer header file **stm32l4xx_ll_ppp.h**
- We have to include them in **main.c** file in USER CODE section.

```
 /* USER CODE BEGIN Includes */
#include "stm32l4xx_ll_adc.h"
#include "stm32l4xx_ll_dac.h"
#include "stm32l4xx_ll_dma.h"
#include "stm32l4xx_ll_gpio.h"
#include "stm32l4xx_ll_rcc.h"
#include "stm32l4xx_ll_tim.h"
#include "stm32l4xx_ll_bus.h"
/* USER CODE END Includes */
```

Replacing HAL functions with init LL

2 - LL_L4_DAC_ADC_Init project – structures declaration

- Each peripheral needs its own initialization structure (**PPP_InitStruct**)
- Some of the peripherals (i.e. timers, ADC, …) need additional one for configuration one of the modes they should be configured in (i.e. Output Capture, Input Capture for timers etc.).
- There is a complete list of needed structures below. Please try to find and declare proper ones.

```
/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
                                                    //GPIO init structure
                                                    //TIM2 init structure
                                                    //TIM2 OC init structure
                                                    //DAC init structure
                                                    //ADC init structure
                                                    //ADC common init structure
                                                    //ADC regular conversion init structure
                                                    //DMA init structure
                                                    //DMA init structure

/* USER CODE END PV */
```

Replacing HAL functions with init LL

2 - LL_L4_DAC_ADC_Init project – structures declaration

- Each peripheral needs its own initialization structure (**PPP_InitStruct**)
- Some of the peripherals (i.e. timers, ADC, …) need additional one for configuration one of the modes they should be configured in (i.e. Output Capture, Input Capture for timers etc.).
- There is a complete list of needed structures below. Please try to find and declare proper ones.

```
 /* USER CODE BEGIN PV */
 /* Private variables ---------------------------------------------------------*/
LL_GPIO_InitTypeDef GPIO_InitStruct;              //GPIO init structure
LL_TIM_InitTypeDef TIM2_InitStruct;               //TIM2 init structure
LL_TIM_OC_InitTypeDef TIM2_OCInitStruct;          //TIM2 OC init structure
LL_DAC_InitTypeDef DAC_InitStruct;                //DAC init structure
LL_ADC_InitTypeDef ADC_InitStruct;                //ADC init structure
LL_ADC_CommonInitTypeDef ADC_ComInitStruct;       //ADC common init structure
LL_ADC_REG_InitTypeDef ADC_RegInitStruct;         //ADC regular conversion init structure
LL_DMA_InitTypeDef DMA_DAC_InitStruct;            //DMA init structure
LL_DMA_InitTypeDef DMA_ADC_InitStruct;            //DMA init structure
 /* USER CODE END PV */
```

Replacing HAL functions with init LL

2 - LL_L4_DAC_ADC_Init project – data buffers declaration

- It is necessary to define the source buffer for DAC (**`dacbuf[]`**) and destination buffer for ADC to store the measured data (**`adcbuf[]`**). Size for both can be **`32`**.

```
/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/









/* USER CODE END PV */
```

Replacing HAL functions with init LL

2 - LL_L4_DAC_ADC_Init project – data buffers declaration

- It is necessary to define the source buffer for DAC (**`dacbuf[]`**) and destination buffer for ADC to store the measured data (**`adcbuf[]`**). Size for both can be **`32`**.

```
/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
#define ADCBUFSIZE  32
#define DACBUFSIZE  32

const uint16_t dacbuf[DACBUFSIZE] = {
                    2047, 2447, 2831, 3185, 3498, 3750, 3939, 4056, 4095, 4056,
                    3939, 3750, 3495, 3185, 2831, 2447, 2047, 1647, 1263, 909,
                    599, 344, 155, 38, 0, 38, 155, 344, 599, 909, 1263, 1647};

uint16_t adcbuf[ADCBUFSIZE];
/* USER CODE END PV */
```

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project – GPIO configuration

The task is to configure 2 analog pins (PA1 – ADC1 Channel6 and PA4 – DAC output1)

1. Before PPP configuration it is necessary to connect the clock to the PPP peripheral. To do this we can use there dedicated macro: **__HAL_RCC_PPP_CLK_ENABLE()**
2. Initialize PPP init structure with default values using **LL_PPP_StructInit()**
3. Fill PPP init structure with required configuration parameters
4. Copy structure fields into PPP (GPIO in this case) registers using **LL_PPP_Init()** function (like in Standard Peripherals Library or in HAL library)
5. Connect GPIO analog switch to ADC input using **LL_GPIO_EnablePinAnalogControl()** function

```
/* USER CODE BEGIN 2 */
/* GPIO LL configuration */
```

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project – GPIO configuration

The task is to configure 2 analog pins (PA1 – ADC1 Channel6 and PA4 – DAC output1)

1. Before PPP configuration it is necessary to connect the clock to the PPP peripheral. To do this we can use there dedicated macro: **__HAL_RCC_PPP_CLK_ENABLE()**
2. Initialize PPP init structure with default values using **LL_PPP_StructInit()**
3. Fill PPP init structure with required configuration parameters
4. Copy structure fields into PPP (GPIO in this case) registers using **LL_PPP_Init()** function (like in Standard Peripherals Library or in HAL library)
5. Connect GPIO analog switch to ADC input using **LL_GPIO_EnablePinAnalogControl()** function

```
/* USER CODE BEGIN 2 */
/* GPIO LL configuration */
__HAL_RCC_GPIOA_CLK_ENABLE();                    //enable clock to the GPIOA peripheral
LL_GPIO_StructInit(&GPIO_InitStruct);            //structure initialization to default values
GPIO_InitStruct.Pin= GPIO_PIN_1 | GPIO_PIN_4;    //set pin 1 (ADC_IN6), 4 (DAC_OUT1)
GPIO_InitStruct.Mode= LL_GPIO_MODE_ANALOG;       //set GPIO as analog mode
GPIO_InitStruct.Pull= LL_GPIO_PULL_NO;           //no pull up or pull down
LL_GPIO_Init(GPIOA,&GPIO_InitStruct);            //initialize GPIOA, pins 1 and 4
LL_GPIO_EnablePinAnalogControl(GPIOA, LL_GPIO_PIN_1);
```

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project – DAC configuration

The task is to configure DAC1, Channel1 to work without output buffer, triggered by Timer2 TRGO signal, without triangle nor noise wave generation

1. Before PPP configuration it is necessary to connect the clock to the PPP peripheral. To do this we can use there dedicated macro: **__HAL_RCC_PPP_CLK_ENABLE()**
2. Initialize PPP init structure with default values using **LL_PPP_StructInit()**
3. Fill PPP init structure with required configuration parameters
4. Copy structure fields into PPP (DAC in this case) registers using **LL_PPP_Init()** function (like in Standard Peripherals Library or in HAL library)

```
/* USER CODE BEGIN 2 */
/* DAC LL configuration */
```

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project – DAC configuration

The task is to configure DAC1, Channel1 to work without output buffer, triggered by Timer2 TRGO signal, without triangle nor noise wave generation

1. Before PPP configuration it is necessary to connect the clock to the PPP peripheral. To do this we can use there dedicated macro: **__HAL_RCC_PPP_CLK_ENABLE()**
2. Initialize PPP init structure with default values using **LL_PPP_StructInit()**
3. Fill PPP init structure with required configuration parameters
4. Copy structure fields into PPP (DAC in this case) registers using **LL_PPP_Init()** function (like in Standard Peripherals Library or in HAL library)

```
/* USER CODE BEGIN 2 */
/* DAC LL configuration */
1 __HAL_RCC_DAC1_CLK_ENABLE(); //enable clock
2 LL_DAC_StructInit(&DAC_InitStruct);
  DAC_InitStruct.TriggerSource          = LL_DAC_TRIG_EXT_TIM2_TRGO;
3 DAC_InitStruct.OutputBuffer           = LL_DAC_OUTPUT_BUFFER_ENABLE;
4 LL_DAC_Init(DAC1, LL_DAC_CHANNEL_1, &DAC_InitStruct);
```

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project - ADC configuration tasks

The task is to configure ADC1, Channel 6 to work in regular continue mode with DMA support, triggered by Timer2 Output Compare event on channel2, with sampling time 12.5 ADC clk cycles. ADC should be clocked by PCLK/2 synchronous clock (40MHz in our case).

1. Before PPP configuration it is necessary to connect the clock to the PPP peripheral. To do this we can use there dedicated macro: **__HAL_RCC_PPP_CLK_ENABLE()**
2. Initialize proper PPP init structure(s) with default values using **LL_PPP_StructInit().** In case of ADC we have 4 different structures:
   a. **LL_ADC_CommonInitTypeDef** – configuration of common parameters for all ADCs (like input clock source) and multimode configuration
   b. **LL_ADC_InitTypeDef** – configuration of the particular ADC basic parameters
   c. **LL_ADC_REG_InitTypeDef** – configuration of the regular conversions
   d. **LL_ADC_INJ_InitTypeDef** – configuration of the injected conversions – NOT USED in the exercise
3. Fill proper PPP init structure(s) with desired configuration parameters
4. Copy structure fields into PPP (ADC in this case) registers using **LL_PPP_Init()** function (like in Standard Peripherals Library or in HAL library)

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project - ADC configuration tasks

The task is to configure ADC1, Channel 6 to work in regular continue mode with DMA support, triggered by Timer2 Output Compare event on channel2, with sampling time 12.5 ADC clk cycles. ADC should be clocked by PCLK/2 synchronous clock (40MHz in our case).

1. At the beginning we should select system clock as clock source for ADC using **LL_RCC_SetADCClockSource()** function
2. Before PPP configuration it is necessary to connect the clock to the ADC1 peripheral. To do this we can use there dedicated macro: **__HAL_RCC_ADC_CLK_ENABLE()**
3. Initialize **LL_ADC_CommonInitTypeDef** structure (defining common settings for all ADCs and multimode configuration) with default values using **LL_ADC_CommonStructInit().**
4. Fill **LL_ADC_CommonInitTypeDef** structure with desired configuration parameters (clock source for ADC)
5. Copy structure fields into ADC registers using **LL_ADC_CommonInit()** function

```
/* ADC LL configuration */
```

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project - ADC configuration tasks

The task is to configure ADC1, Channel 6 to work in regular continue mode with DMA support, triggered by Timer2 Output Compare event on channel2, with sampling time 12.5 ADC clk cycles. ADC should be clocked by PCLK/2 synchronous clock (40MHz in our case).

1. At the beginning we should select system clock as clock source for ADC using **LL_RCC_SetADCClockSource()** function
2. Before PPP configuration it is necessary to connect the clock to the ADC1 peripheral. To do this we can use there dedicated macro: **__HAL_RCC_ADC_CLK_ENABLE()**
3. Initialize **LL_ADC_CommonInitTypeDef** structure (defining common settings for all ADCs and multimode configuration) with default values using **LL_ADC_CommonStructInit().**
4. Fill **LL_ADC_CommonInitTypeDef** structure with desired configuration parameters (clock source for ADC)
5. Copy structure fields into ADC registers using **LL_ADC_CommonInit()** function

```
/* ADC LL configuration */
1 LL_RCC_SetADCClockSource(LL_RCC_ADC_CLKSOURCE_SYSCLK);
2 __HAL_RCC_ADC_CLK_ENABLE();//enable clock
3 LL_ADC_CommonStructInit(&ADC_ComInitStruct);
4 ADC_ComInitStruct.CommonClock = LL_ADC_CLOCK_SYNC_PCLK_DIV2;
5 LL_ADC_CommonInit(ADC1,&ADC_ComInitStruct);
```

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project - ADC configuration tasks

The task is to configure ADC1, Channel 6 to work in regular continue mode with DMA support, triggered by Timer2 Output Compare event on channel2, with sampling time 12.5 ADC clk cycles. ADC should be clocked by PCLK/2 synchronous clock (40MHz in our case).

1. Initialize **LL_ADC_InitTypeDef** structure (defining general settings for particular ADC) with default values using **LL_ADC_StructInit().**
2. Fill **LL_ADC_InitTypeDef** structure with desired configuration parameters (clock source for ADC)
3. Copy structure fields into ADC registers using **LL_ADC_Init()** function

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project - ADC configuration tasks

The task is to configure ADC1, Channel 6 to work in regular continue mode with DMA support, triggered by Timer2 Output Compare event on channel2, with sampling time 12.5 ADC clk cycles. ADC should be clocked by PCLK/2 synchronous clock (40MHz in our case).

1. Initialize **LL_ADC_InitTypeDef** structure (defining general settings for particular ADC) with default values using **LL_ADC_StructInit().**
2. Fill **LL_ADC_InitTypeDef** structure with desired configuration parameters (clock source for ADC)
3. Copy structure fields into ADC registers using **LL_ADC_Init()** function

```
1  LL_ADC_StructInit(&ADC_InitStruct);
   ADC_InitStruct.Resolution = LL_ADC_RESOLUTION_12B ;
2  ADC_InitStruct.DataAlignment = LL_ADC_DATA_ALIGN_RIGHT;
   ADC_InitStruct.LowPowerMode = LL_ADC_LP_MODE_NONE;
3  LL_ADC_Init(ADC1,&ADC_InitStruct);
```

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project - ADC configuration tasks

The task is to configure ADC1, Channel 6 to work in regular continue mode with DMA support, triggered by Timer2 Output Compare event on channel2, with sampling time 12.5 ADC clk cycles. ADC should be clocked by PCLK/2 synchronous clock (40MHz in our case).

1. Initialize **LL_ADC_REG_InitTypeDef** structure (defining regular conversion parameters) with default values using **LL_ADC_REG_StructInit().**
2. Fill **LL_ADC_REG_InitTypeDef** structure with desired configuration parameters (clock source for ADC)
3. Copy structure fields into ADC registers using **LL_ADC_REG_Init()** function

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project - ADC configuration tasks

The task is to configure ADC1, Channel 6 to work in regular continue mode with DMA support, triggered by Timer2 Output Compare event on channel2, with sampling time 12.5 ADC clk cycles. ADC should be clocked by PCLK/2 synchronous clock (40MHz in our case).

1. Initialize **LL_ADC_REG_InitTypeDef** structure (defining regular conversion parameters) with default values using **LL_ADC_REG_StructInit().**
2. Fill **LL_ADC_REG_InitTypeDef** structure with desired configuration parameters (clock source for ADC)
3. Copy structure fields into ADC registers using **LL_ADC_REG_Init()** function

1
```
LL_ADC_REG_StructInit(&ADC_RegInitStruct);
```
2
```
ADC_RegInitStruct.TriggerSource = LL_ADC_REG_TRIG_EXT_TIM2_CH2;
ADC_RegInitStruct.SequencerDiscont = LL_ADC_REG_SEQ_DISCONT_DISABLE;
ADC_RegInitStruct.SequencerLength = LL_ADC_REG_SEQ_SCAN_DISABLE;
ADC_RegInitStruct.ContinuousMode = LL_ADC_REG_CONV_SINGLE;
ADC_RegInitStruct.DMATransfer = LL_ADC_REG_DMA_TRANSFER_UNLIMITED;
ADC_RegInitStruct.Overrun = LL_ADC_REG_OVR_DATA_PRESERVED;
```
3
```
LL_ADC_REG_Init(ADC1,&ADC_RegInitStruct);
```

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project - ADC configuration 3/3

The task is to configure ADC1, Channel 6 to work in regular continue mode with DMA support, triggered by Timer2 Output Compare event on channel2, with sampling time 12.5 ADC clk cycles. ADC should be clocked by PCLK/2 synchronous clock (40MHz in our case).

1. Configure trigger edge to rising using **LL_ADC_REG_SetTriggerEdge()** function
2. Configure the sequencer to:
   a. setting the order of the channels to be converted (in our case single channel 6)
   b. select sampling time for each channel

   There is a single function to set both parameters: **LL_ADC_SetChannelSamplingTime()**
3. After the reset ADC is in deep power down mode. It is necessary to disable this mode using **LL_ADC_DisableDeepPowerDown()** function
4. Further we need to enable ADC internal voltage regulator using **LL_ADC_EnableInternalRegulator()** function and wait for it stabilization (implement your own delay() function)

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project - ADC configuration 3/3

The task is to configure ADC1, Channel 6 to work in regular continue mode with DMA support, triggered by Timer2 Output Compare event on channel2, with sampling time 12.5 ADC clk cycles. ADC should be clocked by PCLK/2 synchronous clock (40MHz in our case).

1. Configure trigger edge to rising using **LL_ADC_REG_SetTriggerEdge()** function
2. Configure the sequencer to:
   a. setting the order of the channels to be converted (in our case single channel 6)
   b. select sampling time for each channel

   There is a single function to set both parameters: **LL_ADC_SetChannelSamplingTime()**
3. After the reset ADC is in deep power down mode. It is necessary to disable this mode using **LL_ADC_DisableDeepPowerDown()** function
4. Further we need to enable ADC internal voltage regulator using **LL_ADC_EnableInternalRegulator()** function and wait for it stabilization (implement your own delay() function)

```
1   LL_ADC_REG_SetTriggerEdge(ADC1,LL_ADC_REG_TRIG_EXT_RISING);
2a  LL_ADC_REG_SetSequencerRanks(ADC1,LL_ADC_REG_RANK_1,LL_ADC_CHANNEL_6);
2b  LL_ADC_SetChannelSamplingTime(ADC1, LL_ADC_CHANNEL_6 , LL_ADC_SAMPLINGTIME_12CYCLES_5);

3   LL_ADC_DisableDeepPowerDown(ADC1);
4   LL_ADC_EnableInternalRegulator(ADC1);
      //wait 20us for internal regulator stabilization
```

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project - DAC_DMA configuration

The task is to configure Channel 3 in DMA1 to work with DAC using **LL_DMA_InitTypeDef** structure and **LL_DMA_Init()** function in the following way:

- Continuously read data from internal buffer (dacbuf[DACBUFSIZE]) in halfwords, with pointer incrementation
- Continuously write data (in halfwords, without pointer incrementation) to DAC1 data register for channel1 (12bits, right alignment)*.
- After the configuration we should enable the Channel3 in DMA1 using **LL_DMA_EnableChannel()** function

*) Hint: To get the proper address we can use function **LL_DAC_DMA_GetRegAddr()**

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project - DAC_DMA configuration

The task is to configure Channel 3 in DMA1 to work with DAC using **LL_DMA_InitTypeDef** structure and **LL_DMA_Init()** function in the following way:

- Continuously read data from internal buffer (dacbuf[DACBUFSIZE]) in halfwords, with pointer incrementation
- Continuously write data (in halfwords, without pointer incrementation) to DAC1 data register for channel1 (12bits, right alignment)*.
- After the configuration we should enable the Channel3 in DMA1 using **LL_DMA_EnableChannel()** function

```
/* DAC DMA LL configuration */
__HAL_RCC_DMA1_CLK_ENABLE(); //enable clock
LL_DMA_StructInit(&DMA_DAC_InitStruct);
DMA_DAC_InitStruct.PeriphOrM2MSrcAddress = LL_DAC_DMA_GetRegAddr(DAC1,LL_DAC_CHANNEL_1,LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED);
DMA_DAC_InitStruct.MemoryOrM2MDstAddress = (uint32_t)dacbuf;
DMA_DAC_InitStruct.Direction = LL_DMA_DIRECTION_MEMORY_TO_PERIPH;
DMA_DAC_InitStruct.Mode = LL_DMA_MODE_CIRCULAR;
DMA_DAC_InitStruct.PeriphOrM2MSrcIncMode = LL_DMA_PERIPH_NOINCREMENT;
DMA_DAC_InitStruct.MemoryOrM2MDstIncMode = LL_DMA_MEMORY_INCREMENT;
DMA_DAC_InitStruct.PeriphOrM2MSrcDataSize = LL_DMA_PDATAALIGN_HALFWORD;
DMA_DAC_InitStruct.MemoryOrM2MDstDataSize = LL_DMA_MDATAALIGN_HALFWORD;
DMA_DAC_InitStruct.NbData = DACBUFSIZE;
DMA_DAC_InitStruct.PeriphRequest = LL_DMA_REQUEST_6;
DMA_DAC_InitStruct.Priority = LL_DMA_PRIORITY_LOW;
LL_DMA_Init(DMA1, LL_DMA_CHANNEL_3, &DMA_DAC_InitStruct);
LL_DMA_EnableChannel(DMA1, LL_DMA_CHANNEL_3);
```

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project - ADC_DMA configuration

The task is to configure Channel 1 in DMA1 to work with ADC using **LL_DMA_InitTypeDef** structure and **LL_DMA_Init()** function in the following way:

- Continuously read data (in halfwords, without pointer incrementation) from ADC1 regular data register*.
- Continuously write data to internal buffer (adcbuf[ADCBUFSIZE]) in halfwords, with pointer incrementation
- After the configuration we should enable the Channel1 in DMA1 using **LL_DMA_EnableChannel()** function

*) Hint: To get the proper address we can use function **LL_ADC_DMA_GetRegAddr()**

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project - ADC_DMA configuration

The task is to configure Channel 1 in DMA1 to work with ADC using **LL_DMA_InitTypeDef** structure and **LL_DMA_Init()** function in the following way:

- Continuously read data (in halfwords, without pointer incrementation) from ADC1 regular data register*.
- Continuously write data to internal buffer (adcbuf[ADCBUFSIZE]) in halfwords, with pointer incrementation
- After the configuration we should enable the Channel1 in DMA1 using **LL_DMA_EnableChannel()** function

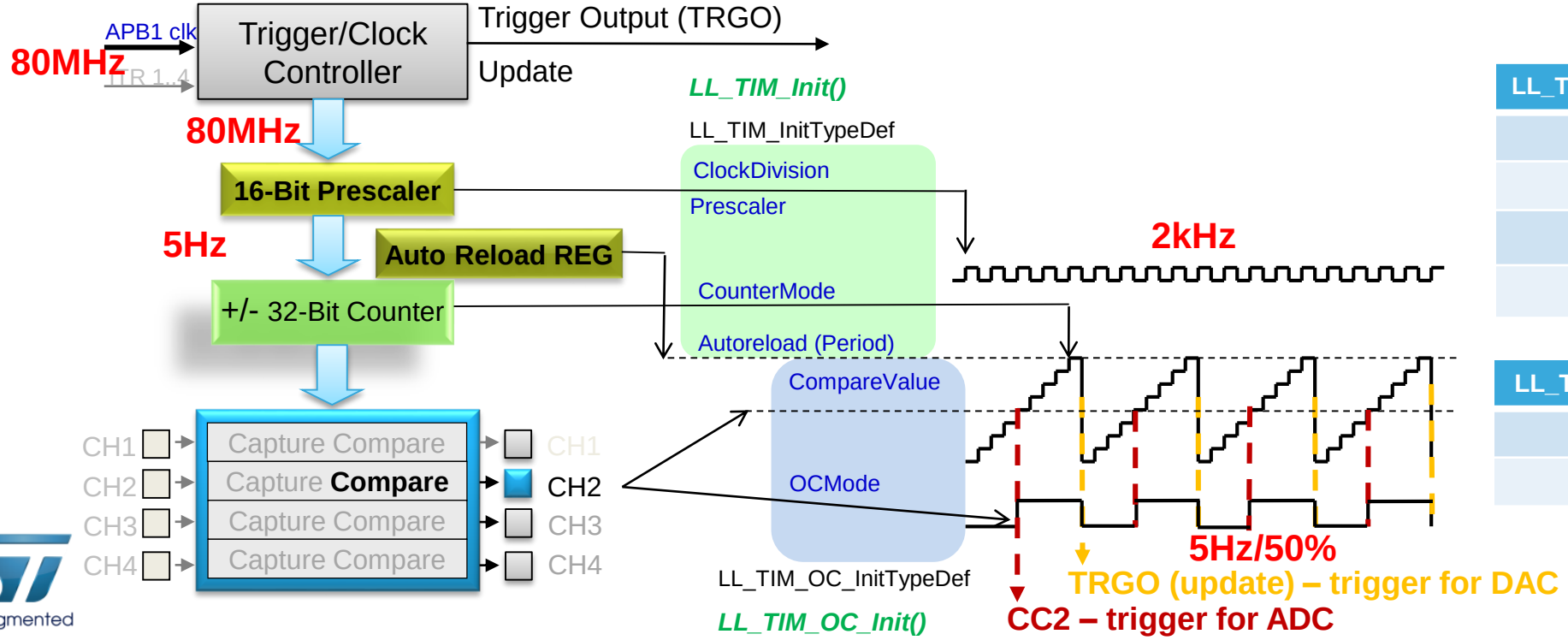
```
/* ADC DMA LL configuration */
  LL_DMA_StructInit(&DMA_ADC_InitStruct);
  DMA_ADC_InitStruct.PeriphOrM2MSrcAddress = LL_ADC_DMA_GetRegAddr(ADC1,LL_ADC_DMA_REG_REGULAR_DATA);
  DMA_ADC_InitStruct.MemoryOrM2MDstAddress = (uint32_t)adcbuf;
  DMA_ADC_InitStruct.Direction = LL_DMA_DIRECTION_PERIPH_TO_MEMORY;
  DMA_ADC_InitStruct.Mode = LL_DMA_MODE_CIRCULAR;
  DMA_ADC_InitStruct.PeriphOrM2MSrcIncMode = LL_DMA_PERIPH_NOINCREMENT;
  DMA_ADC_InitStruct.MemoryOrM2MDstIncMode = LL_DMA_MEMORY_INCREMENT;
  DMA_ADC_InitStruct.PeriphOrM2MSrcDataSize = LL_DMA_PDATAALIGN_HALFWORD;
  DMA_ADC_InitStruct.MemoryOrM2MDstDataSize = LL_DMA_MDATAALIGN_HALFWORD;
  DMA_ADC_InitStruct.NbData = ADCBUFSIZE;
  DMA_ADC_InitStruct.PeriphRequest = LL_DMA_REQUEST_0;
  DMA_ADC_InitStruct.Priority = LL_DMA_PRIORITY_LOW;
  LL_DMA_Init(DMA1, LL_DMA_CHANNEL_1, &DMA_ADC_InitStruct);
  LL_DMA_EnableChannel(DMA1, LL_DMA_CHANNEL_1);
```

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project – TIM2 configuration

The task is to configure Timer2 (TIM2) to work in up-counting mode with the following parameters/options:

- Its output compare channel 2 should be configured in toggle mode with output compare parameters: frequency 5Hz, duty cycle 50%.
- There is no need to output Channel2 on the pin.
- TRGO signal of Timer2 should be configured on update and it will trigger DAC conversions
- Compare event on channel 2 will be used to trigger the ADC conversions.
- Source clock for the timer is APB clock = 80MHz



LL_TIM_InitTypeDef	value
ClockDivision	?
Prescaler	?
CounterMode	?
Autoreload	?

LL_TIM_OC_InitTypeDef	value
CompareValue	?
OCMode	?

life.augmented

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project – TIM2 configuration

Configuration of the clock and counter block of the timer (LL_TIM_InitTypeDef structure)

1. Connect clock to Timer2 (TIM2) using macro **__HAL_RCC_TIM2_CLK_ENABLE()**
2. Initialize **LL_TIM_InitTypeDef** structure with default values using **LL_TIM_StructInit()** function**.**
3. Fill **LL_TIM_InitTypeDef** structure with desired configuration parameters (calculated clock prescalers, mode and autoreload value for TIM2 to reach target 5Hz, 50% dc on OC channel4)
4. Copy structure fields into TIM2 registers using **LL_TIM_Init()** function

```
/* TIM2 LL configuration */
```

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project – TIM2 configuration

Configuration of the clock and counter block of the timer (LL_TIM_InitTypeDef structure)

1. Connect clock to Timer2 (TIM2) using macro **__HAL_RCC_TIM2_CLK_ENABLE()**
2. Initialize **LL_TIM_InitTypeDef** structure with default values using **LL_TIM_StructInit()** function**.**
3. Fill **LL_TIM_InitTypeDef** structure with desired configuration parameters (calculated clock prescalers, mode and autoreload value for TIM2 to reach target 5Hz, 50% dc on OC channel4)
4. Copy structure fields into TIM2 registers using **LL_TIM_Init()** function

```
/* TIM2 LL configuration */
__HAL_RCC_TIM2_CLK_ENABLE(); //enable clock
LL_TIM_StructInit(&TIM2_InitStruct);
TIM2_InitStruct.Prescaler = 39999;
TIM2_InitStruct.CounterMode = LL_TIM_COUNTERMODE_UP;
TIM2_InitStruct.Autoreload = 399;
TIM2_InitStruct.ClockDivision = LL_TIM_CLOCKDIVISION_DIV1;
LL_TIM_Init(TIM2,&TIM2_InitStruct);
```

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project – TIM2 configuration

Configuration of the output compare block for channel2 of the timer2 (LL_TIM_OCInitTypeDef structure)

1. Initialize **LL_TIM_OCInitTypeDef** structure with default values using **LL_TIM_OCStructInit()** function.
2. Fill **LL_TIM_OCInitTypeDef** structure with desired configuration parameters (calculated pulse, mode and autoreload value for TIM2 to reach target 5Hz, 50% dc on OC channel4)
3. Copy structure fields into TIM2 registers using **LL_TIM_OC_Init()** function
4. Configure trigger output (TRGO) to update using **LL_TIM_SetTriggerOutput()** function.
5. Enable TIM2 channel2 using **LL_TIM_CC_EnableChannel()** function.

```
/* TIM2 output compare mode for channel 2 - LL configuration */
```

Replacing HAL functions with init LL

3 - LL_L4_DAC_ADC_Init project – TIM2 configuration

Configuration of the output compare block for channel2 of the timer2 (LL_TIM_OCInitTypeDef structure)

1. Initialize **LL_TIM_OCInitTypeDef** structure with default values using **LL_TIM_OCStructInit()** function.
2. Fill **LL_TIM_OCInitTypeDef** structure with desired configuration parameters (calculated pulse, mode and autoreload value for TIM2 to reach target 5Hz, 50% dc on OC channel4)
3. Copy structure fields into TIM2 registers using **LL_TIM_OC_Init()** function
4. Configure trigger output (TRGO) to update using **LL_TIM_SetTriggerOutput()** function.
5. Enable TIM2 channel2 using **LL_TIM_CC_EnableChannel()** function.

```
/* TIM2 output compare mode for channel 2 - LL configuration */
1 LL_TIM_OC_StructInit(&TIM2_OCInitStruct);
  TIM2_OCInitStruct.OCMode = LL_TIM_OCMODE_TOGGLE;
  TIM2_OCInitStruct.OCState = LL_TIM_OCSTATE_ENABLE;
2 TIM2_OCInitStruct.CompareValue = 200;
  TIM2_OCInitStruct.OCPolarity = LL_TIM_OCPOLARITY_HIGH;
3 LL_TIM_OC_Init(TIM2,LL_TIM_CHANNEL_CH2,&TIM2_OCInitStruct);
4 LL_TIM_SetTriggerOutput(TIM2, LL_TIM_TRGO_UPDATE);
5 LL_TIM_CC_EnableChannel(TIM2, LL_TIM_CHANNEL_CH2);
```

Replacing HAL functions with init LL

4 - LL_L4_DAC_ADC_Init project – starting the peripherals

Start already configured peripherals:

1. Enable DMA for Channel1 of DAC1 using **LL_DAC_EnableDMAReq()** function
2. Enable trigger for Channel1 of DAC1 using **LL_DAC_EnableTrigger()** function
3. Enable Channel1 of DAC1 using **LL_DAC_Enable()** function
4. Start calibration of ADC1 (for single ended conversions) using function **LL_ADC_StartCalibration()**.
5. **Add necessary 116 ADC clk delay after calibration start**
6. Enable ADC1 using **LL_ADC_Enable()** function
7. Start regular conversion (ADC1 will start conversion after next HW trigger) using **LL_ADC_REG_StartConversion()** function
8. Activate timer2 using **LL_TIM_EnableCounter()** function

```
/* DAC activation */



/* ADC activation */



/* TIM2 activation */
```

Replacing HAL functions with init LL

4 - LL_L4_DAC_ADC_Init project – starting the peripherals

Start already configured peripherals:

1. Enable DMA for Channel1 of DAC1 using **LL_DAC_EnableDMAReq()** function
2. Enable trigger for Channel1 of DAC1 using **LL_DAC_EnableTrigger()** function
3. Enable Channel1 of DAC1 using **LL_DAC_Enable()** function
4. Start calibration of ADC1 (for single ended conversions) using function **LL_ADC_StartCalibration()**.
5. **Add necessary 116 ADC clk delay after calibration start**
6. Enable ADC1 using **LL_ADC_Enable()** function
7. Start regular conversion (ADC1 will start conversion after next HW trigger) using **LL_ADC_REG_StartConversion()** function
8. Activate timer2 using **LL_TIM_EnableCounter()** function

```
/* DAC activation */
1 LL_DAC_EnableDMAReq(DAC1, LL_DAC_CHANNEL_1);
2 LL_DAC_EnableTrigger(DAC1, LL_DAC_CHANNEL_1);
3 LL_DAC_Enable(DAC1, LL_DAC_CHANNEL_1);

/* ADC activation */
4 LL_ADC_StartCalibration(ADC1, LL_ADC_SINGLE_ENDED);
5 //necessary 116 ADC clk delay
6 LL_ADC_Enable(ADC1);
7 LL_ADC_REG_StartConversion(ADC1);

/* TIM2 activation */
8 LL_TIM_EnableCounter(TIM2);
```

Comparison for L4_DAC_ADC example

Role	HAL		Init Low Layer Library (based on structures)			Unitary Low Layer Library (based on simple functions)			
	function name	size [B]	function name	size [B]	Reduction vs HAL	function name	size [B]	Reduction vs LL structures	Reduction vs HAL
Startup & Initialization	HAL_Init	1432	HAL_Init	1256	**12%**	HAL_Init	1256	**0%**	**12%**
Clock configuration	SystemClock_Config	5512	LL_SystemClock_Config	1552	**72%**	LL_SystemClock_Config	1552	**0%**	**72%**
GPIO configuration	MX_GPIO_Init	5552	GPIO_LL_configuration	1984	**64%**	LL_GPIO_configuration	1680	**15%**	**70%**
ADC configuration	MX_ADC1_Init	8400	ADC_LL_configuration	2480	**70%**	LL_ADC_configuration	1896	**24%**	**77%**
DAC configuration	MX_DAC1_Init	8896	DAC_LL_configuration	2672	**70%**	LL_DAC_configuration	1960	**27%**	**78%**
ADC DMA configuration	MX_DMA_Init	8984	DMA_ADC_LL_configuration	2928	**na**	LL_DMA_ADC_configuration	2080	**29%**	**na**
DAC DMA configuration			DMA_DAC_LL_configuration	3120	**65%**	LL_DMA_DAC_configuration	2288	**27%**	**75%**
TIM2 configuration	MX_TIM2_Init	10776	TIM2_LL_configuration	4272	**60%**	LL_TIM2_configuration	2400	**44%**	**78%**
ADC activation	HAL_ADC_Start_DMA	11424	ADC_LL_activation	4344	**62%**	ADC_LL_activation	2472	**43%**	**78%**
DAC activation	HAL_DAC_Start_DMA	11816	DAC_LL_activation	4384	**63%**	DAC_LL_activation	2512	**43%**	**79%**
TIM2 activation	HAL_TIM_OC_Start	11936	TIM2_LL_activation	4400	**63%**	TIM2_LL_activation	2528	**43%**	**79%**
ADC without calibration									

Complete application code size

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