STM32 Ecosystem workshop

T.O.M.A.S Team

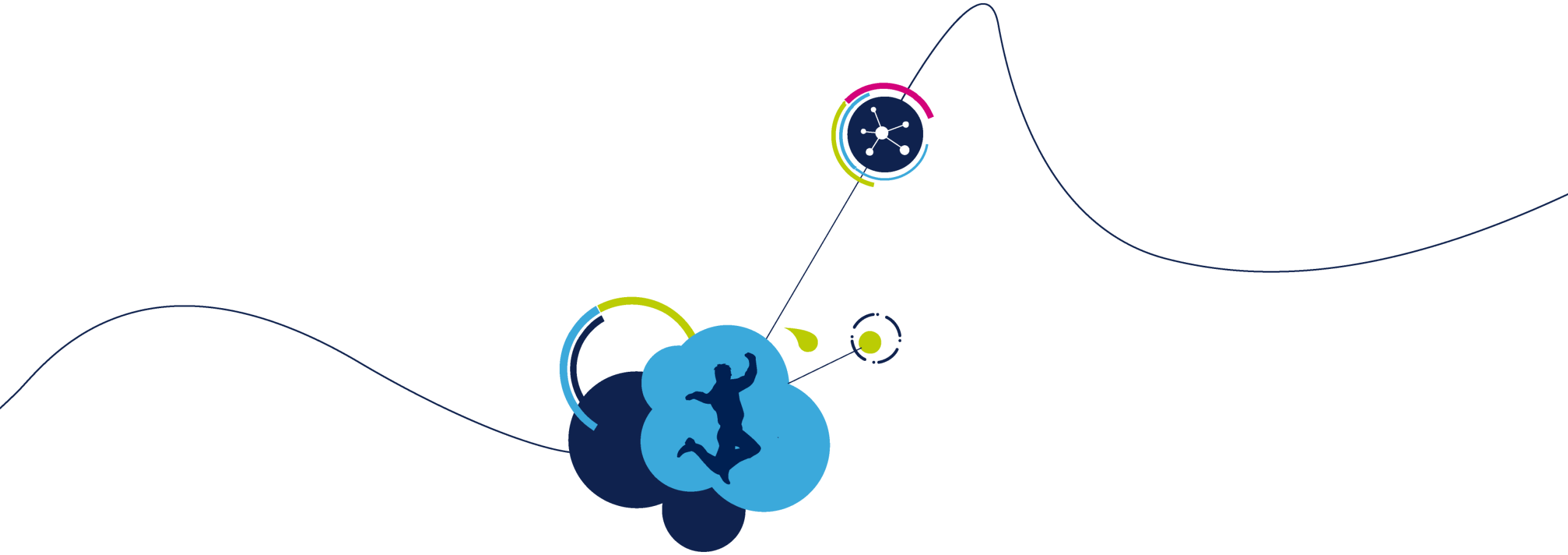
life.augmented

We will continue a bit more with software activities.

- Let’s try to rewrite our L4_DAC_ADC application to use 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.
- There are two ways to complete this task:
  - Use of unitary init functions requiring good knowledge of the peripherals (following reference manual configuration steps)
  - Use of init functions (similar method to Standard Peripherals Library)
- Let’s try the first option, then we can compare the code size of the projects
- What would be the difference?

Goal of this part

- ✓Gain knowledge about complete ST software offer for STM32 microcontrollers
- ✓Gain knowledge about Low Layer Library concepts: unitary and init
- ❑Practice Low Layer Library concept on previously generated HAL based project
- ❑Gain knowledge about differences between HAL and LL concepts.

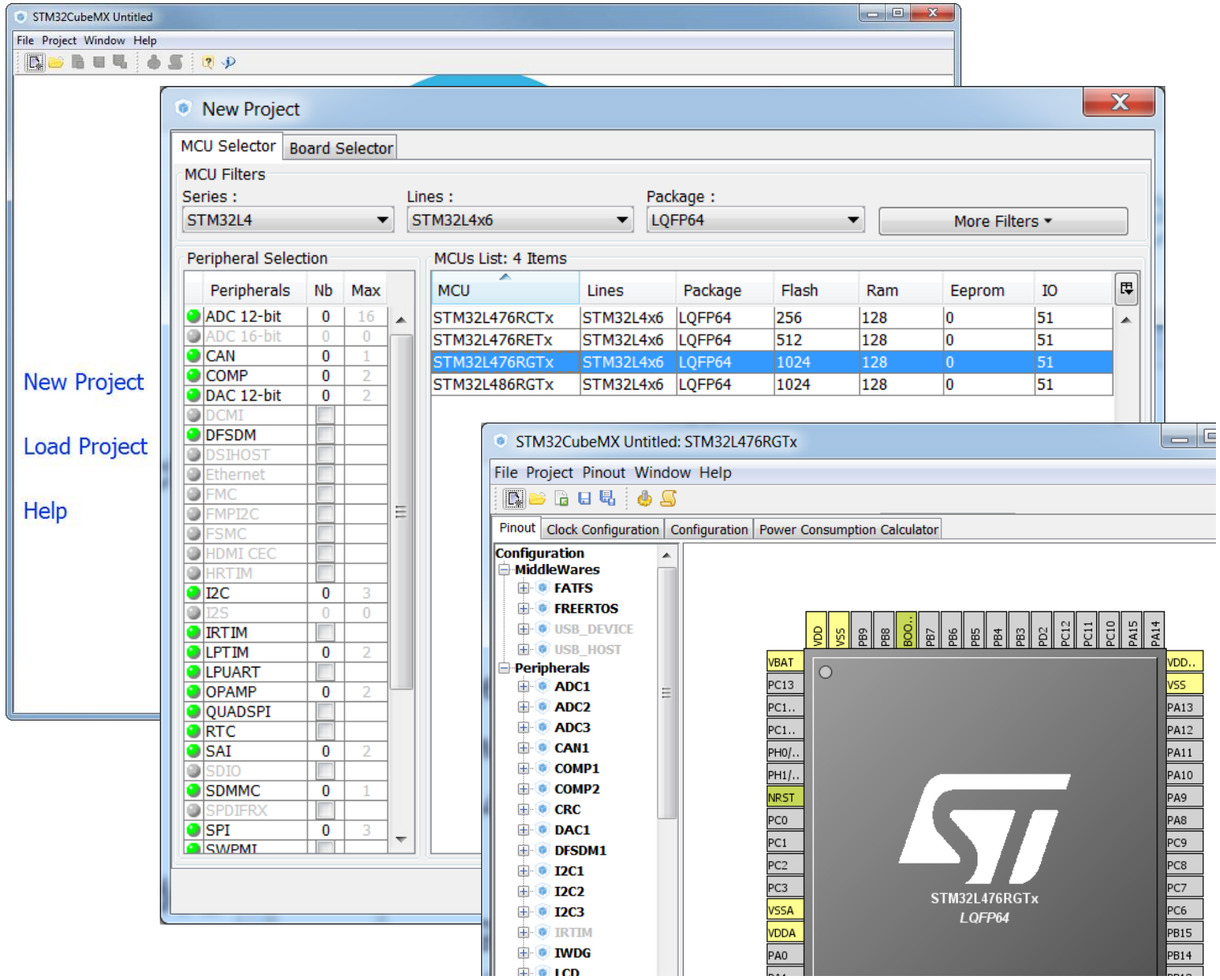


Creating the L4_DAC_ADC project with full usage of LL library

life.augmented

New project creation

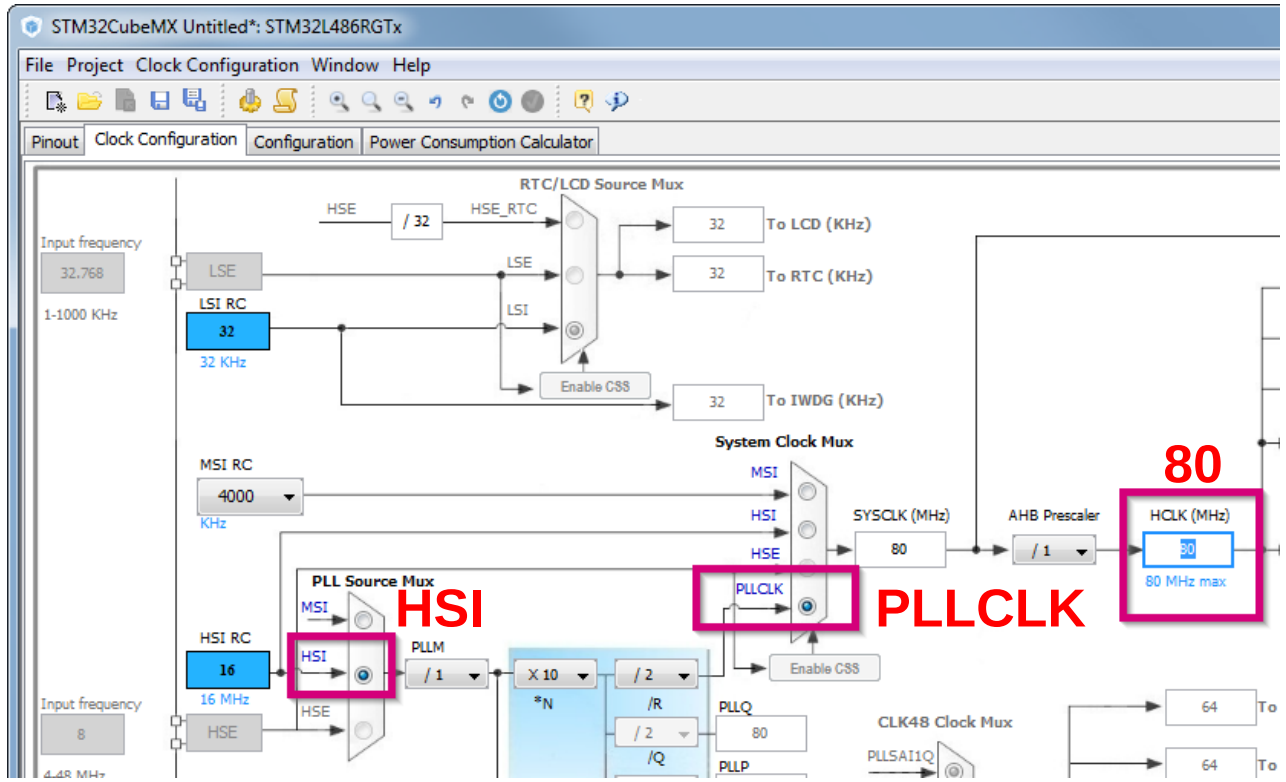
- Open STM32CubeMX
- Select **New Project**
- *Menu → File → New project*
- Select STM32L4
  - STM32L4x6
  - LQFP64 package
  - STM32L476RGTx
- Do not select any peripherals (will be added manually later on)

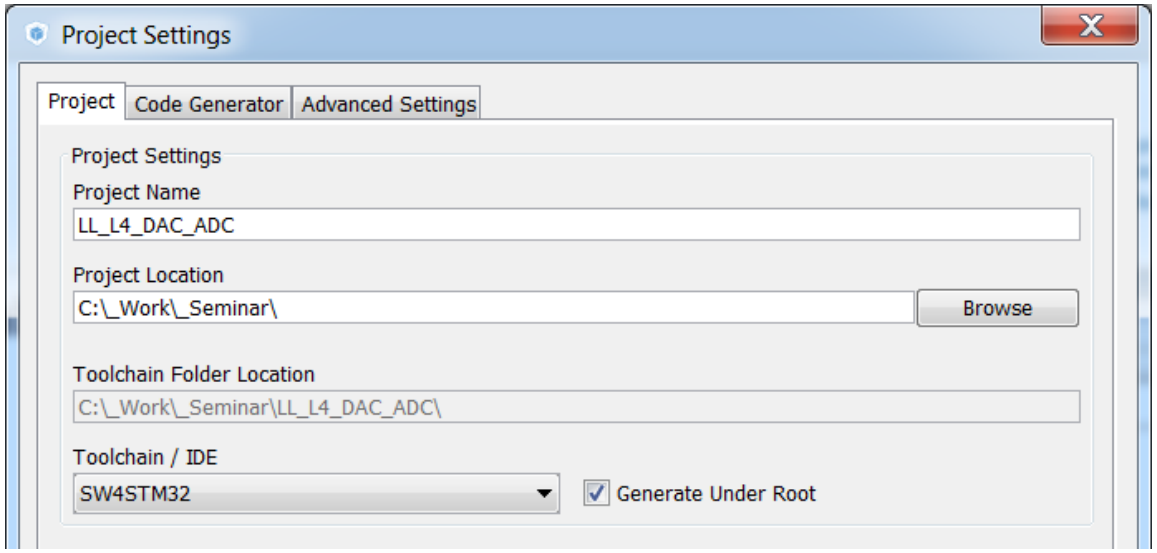


life.augmented

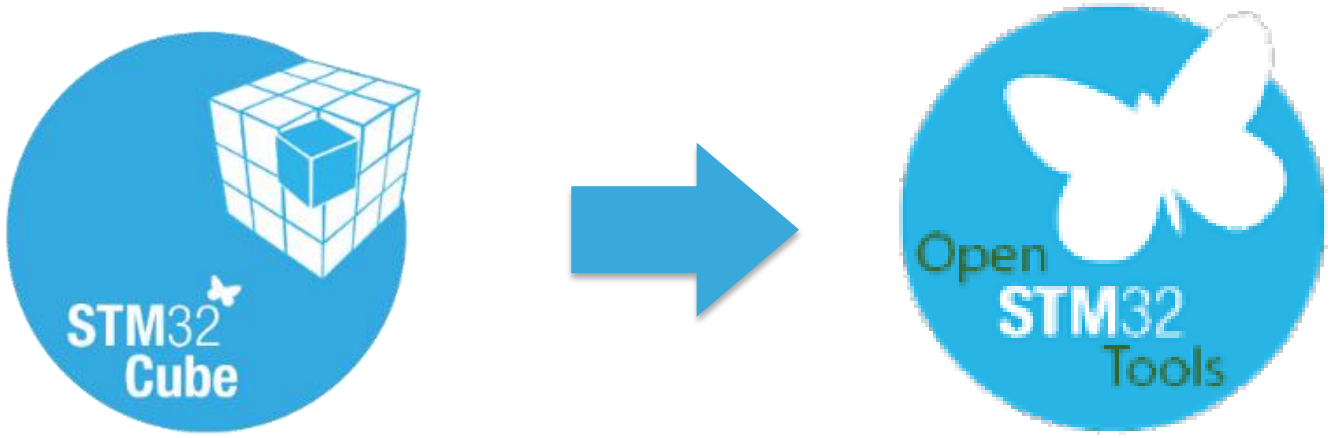
Template project generation

- Go to **Clock Configuration** Tab
- Set clocks to 80MHz (based on 16MHz HSI*PLL)
- Save project as **LL_L4_DAC_ADC**
- Generate the C project for SW4STM32
- Import new project into the SW4STM32 workspace used in previous exercises



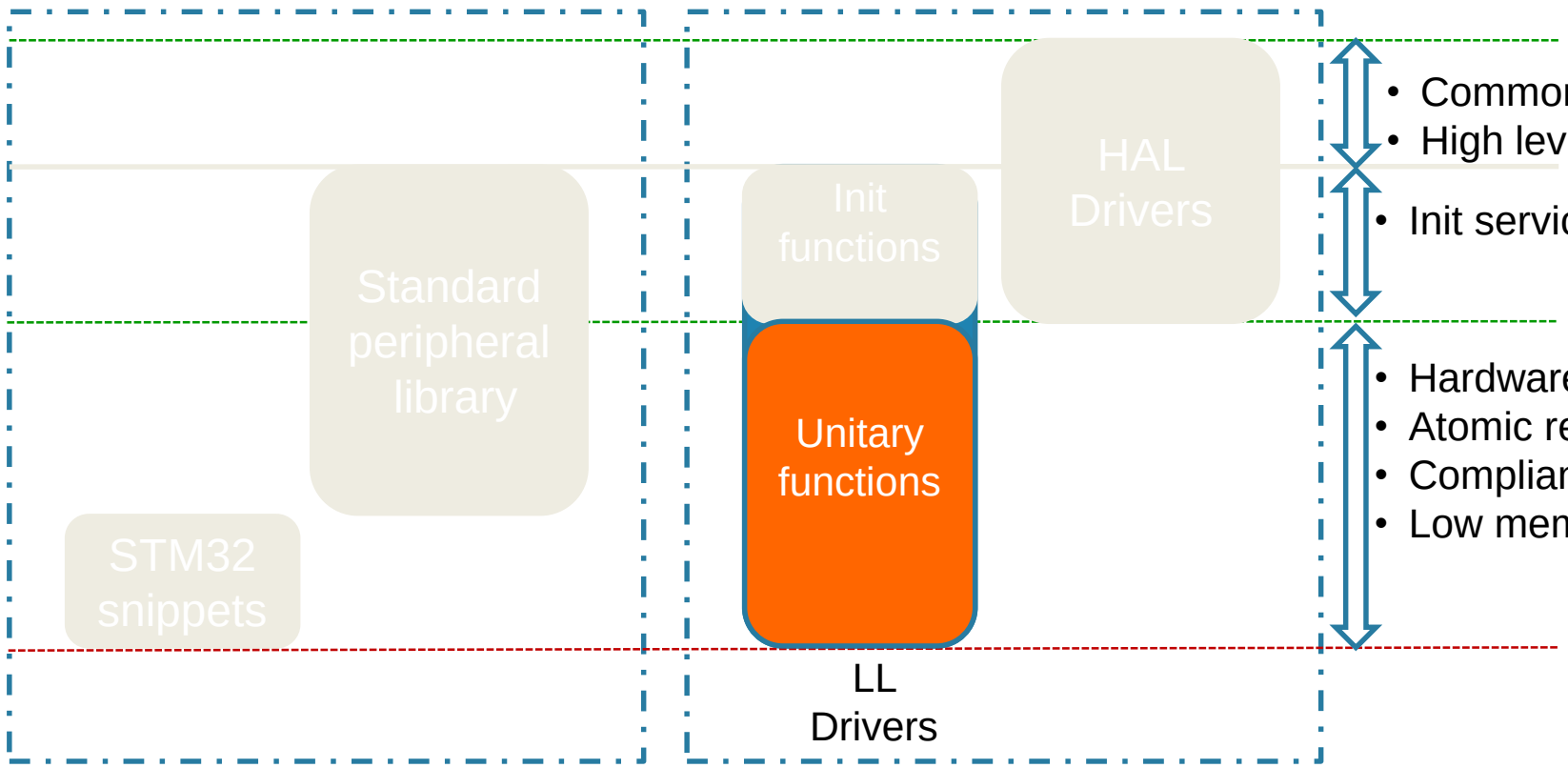


- Generate the code with new features added
- Open newly generated project in SW4STM32



LL_L4_DAC_ADC

using unitary functions



- Common services (portability)
- High level state machine based processes

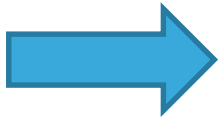
- Init services

- Hardware basic services
- Atomic register access
- Compliancy with the HAL drivers
- Low memory footprint

Copy LL library files into the project

- From repository ***STM32CubeMX→Menu→Help→Updater Settings*** get repository path (marked **.** below)
- 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

Writing the code

LL_L4_DAC_ADC project - tasks

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

1. Include necessary LL header files.
2. Declare data 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 LL functions.

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

Replacing HAL functions with unitary LL

1 - LL_L4_DAC_ADC 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.
- Please try to find and declare proper ones:

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

Replacing HAL functions with unitary LL

1 - LL_L4_DAC_ADC 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 unitary LL

2 - LL_L4_DAC_ADC 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 unitary LL

2 - LL_L4_DAC_ADC 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 unitary LL

3 - LL_L4_DAC_ADC project – GPIO configuration

The task is to configure 2 analog pins (PA1 – ADC1 Channel6 and PA4 – DAC1 output1)

1. Before PPP configuration it is necessary to connect the clock to the PPP peripheral.
   To do this we can use dedicated function: **LL_AHB2_GRP1_EnableClock()**
2. Configure GPIOA, pin1 into analog mode using **LL_GPIO_SetPinMode()**
3. Perform step 2 for GPIOA, pin4
4. Connect GPIO analog switch to ADC1 input for PA1 using **LL_GPIO_EnablePinAnalogControl()**

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

life.augmented

Use stm32l4xx_ll_gpio.h as a reference

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project – GPIO configuration

The task is to configure 2 analog pins (PA1 – ADC1 Channel6 and PA4 – DAC1 output1)

1. Before PPP configuration it is necessary to connect the clock to the PPP peripheral.
   To do this we can use dedicated function: **LL_AHB2_GRP1_EnableClock()**
2. Configure GPIOA, pin1 into analog mode using **LL_GPIO_SetPinMode()**
3. Perform step 2 for GPIOA, pin4
4. Connect GPIO analog switch to ADC1 input for PA1 using **LL_GPIO_EnablePinAnalogControl()**

```
/* USER CODE BEGIN 2 */
/* GPIO LL configuration */
1 LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_GPIOA); //enable clock to the GPIOA peripheral
2 LL_GPIO_SetPinMode(GPIOA, LL_GPIO_PIN_1, LL_GPIO_MODE_ANALOG);
3 LL_GPIO_SetPinMode(GPIOA, LL_GPIO_PIN_4, LL_GPIO_MODE_ANALOG);
4 LL_GPIO_EnablePinAnalogControl(GPIOA, LL_GPIO_PIN_1);
```

Time for homework ;)

Let’s stop at this point – you can find full description of further steps in the presentation.

Please merge code:

- from ***template_src.c*** file (lines 138,139 and 159-260)
- into ***main.c*** file after LL_GPIO_EnablePinAnalogControl() function.

Compile the code and compare the code size between HAL and LL version of the same application.

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project – DAC configuration

The task is to configure DAC1, Channel1 to work with 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 dedicated macro: **LL_APB1_GRP1_EnableClock()**
2. Select trigger source (TRGO signal from TIM2) using **LL_DAC_SetTriggerSource()**
3. Configure DAC1 output for Channel1 in normal mode (no sample and hold usage) with buffer enable and connection to GPIO (PA4 in our case) using **LL_DAC_ConfigOutput()**
4. Enable DMA requests for DAC1, channel1 using **LL_DAC_EnableDMAReq()**

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


Use stm32l4xx_ll_dac.h as a reference

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project – DAC configuration

The task is to configure DAC1, Channel1 to work with 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 dedicated macro: **LL_APB1_GRP1_EnableClock()**
2. Select trigger source (TRGO signal from TIM2) using **LL_DAC_SetTriggerSource()**
3. Configure DAC1 output for Channel1 in normal mode (no sample and hold usage) with buffer enable and connection to GPIO (PA4 in our case) using **LL_DAC_ConfigOutput()**
4. Enable DMA requests for DAC1, channel1 using **LL_DAC_EnableDMAReq()**

```
/* USER CODE BEGIN 2 */
/* DAC LL configuration */
1 LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_DAC1); //enable clock
2 LL_DAC_SetTriggerSource(DAC1, LL_DAC_CHANNEL_1, LL_DAC_TRIG_EXT_TIM2_TRGO);
3 LL_DAC_ConfigOutput(DAC1, LL_DAC_CHANNEL_1, LL_DAC_OUTPUT_MODE_NORMAL,
          LL_DAC_OUTPUT_BUFFER_ENABLE, LL_DAC_OUTPUT_CONNECT_GPIO);
4 LL_DAC_EnableDMAReq(DAC1, LL_DAC_CHANNEL_1);
```

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project - ADC configuration tasks

The task is to configure ADC1, Channel 6 (PA1) to work in regular single 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 (system clock – synchronous mode or HSI clock – asynchronous mode). Point 6.3.3 in reference manual.
2. Before PPP configuration it is necessary to connect the clock to the PPP peripheral.
3. In the next step we need to configure the input clock for ADC (PCLK/2 synchronous mode)
4. The Next step is to select trigger source for ADC1 regular conversions (capture compare channel2 in timer2)
5. Further we need to configure the trigger signal edge (rising in our case)
6. Further we need to configure single conversion per trigger event
7. Further we need to configure DMA data transfer to unlimited mode
8. In the next steps we need to configure ADC sequencer for selected channel: Point 16.3.11 and further in reference manual.
    a. Configure length of regular sequence (1 in our case)
    b. Configure sequencer ranks for each channel (channel6 only in our case)
    c. Configure sampling time for each channel (channel6 only in our case)
9. After the reset ADC is in deep power down mode. It is necessary to disable this mode. Point 16.3.6 in reference manual.
10. Further we need to enable ADC internal voltage regulator and wait for its stabilization (20us). Point 6.3.17 (table 63) in datasheet

Reference manual and Datasheet are present in Documentation folder within User_Material.zip file

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project - ADC configuration tasks 1/2

The task is to configure ADC1, Channel 6 (PA1) to work in regular single 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 PPP peripheral.
   To do this we can use dedicated macro: **LL_AHB2_GRP1_EnableClock()**
3. In the next step we need to configure the input clock for ADC (PCLK/2 synchronous mode) using **LL_ADC_SetCommonClock()** function
4. The Next step is to select trigger source for ADC1 regular conversions (capture compare channel2 in timer2) using **LL_ADC_REG_SetTriggerSource()** function
5. Further we need to configure the trigger signal edge (rising in our case) using **LL_ADC_REG_SetTriggerEdge()** function
6. Further we need to configure single conversion per trigger event using **LL_ADC_REG_SetContinuousMode()** function

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

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project - ADC configuration tasks 1/2

The task is to configure ADC1, Channel 6 (PA1) to work in regular single 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 PPP peripheral.
   To do this we can use dedicated macro: **LL_AHB2_GRP1_EnableClock()**
3. In the next step we need to configure the input clock for ADC (PCLK/2 synchronous mode) using **LL_ADC_SetCommonClock()** function
4. The Next step is to select trigger source for ADC1 regular conversions (capture compare channel2 in timer2) using **LL_ADC_REG_SetTriggerSource()** function
5. Further we need to configure the trigger signal edge (rising in our case) using **LL_ADC_REG_SetTriggerEdge()** function
6. Further we need to configure single conversion per trigger event using **LL_ADC_REG_SetContinuousMode()** function

```
/* ADC LL configuration */
1 LL_RCC_SetADCClockSource(LL_RCC_ADC_CLKSOURCE_SYSCLK);
2 LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_ADC);//enable clock
3 LL_ADC_SetCommonClock(__LL_ADC_COMMON_INSTANCE(ADC1), LL_ADC_CLOCK_SYNC_PCLK_DIV2);
4 LL_ADC_REG_SetTriggerSource(ADC1, LL_ADC_REG_TRIG_EXT_TIM2_CC2);
5 LL_ADC_REG_SetTriggerEdge(ADC1, LL_ADC_REG_TRIG_EXT_RISING);
6 LL_ADC_REG_SetContinuousMode(ADC1, LL_ADC_REG_CONV_SINGLE);
```

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project - ADC configuration tasks 2/2

The task is to configure ADC1, Channel 6 (PA1) to work in regular single 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).

7. Further we need to configure DMA data transfer to unlimited mode using **LL_ADC_REG_SetDMATransfer()** function
8. Further we need to configure ADC sequencer for regular conversions:
   a. set ADC group regular sequencer length and scan direction using **LL_ADC_REG_SetSequencerLength()** function
   b. set ADC group regular sequence: channel on the selected sequence rank using **LL_ADC_REG_SetSequencerRanks()** function
   c. configure sampling time for given ADC channel using **LL_ADC_SetChannelSamplingTime()** function
9. After the reset ADC is in deep power down mode. It is necessary to disable this mode using **LL_ADC_DisableDeepPowerDown()** function
10. 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 unitary LL

3 - LL_L4_DAC_ADC project - ADC configuration tasks 2/2

The task is to configure ADC1, Channel 6 (PA1) to work in regular single 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).

7. Further we need to configure DMA data transfer to unlimited mode using **LL_ADC_REG_SetDMATransfer()** function
8. Further we need to configure ADC sequencer for regular conversions:
   a. set ADC group regular sequencer length and scan direction using **LL_ADC_REG_SetSequencerLength()** function
   b. set ADC group regular sequence: channel on the selected sequence rank using **LL_ADC_REG_SetSequencerRanks()** function
   c. configure sampling time for given ADC channel using **LL_ADC_SetChannelSamplingTime()** function
9. After the reset ADC is in deep power down mode. It is necessary to disable this mode using **LL_ADC_DisableDeepPowerDown()** function
10. Further we need to enable ADC internal voltage regulator using **LL_ADC_EnableInternalRegulator()** function and wait for it stabilization (implement your own delay() function)

```
7   LL_ADC_REG_SetDMATransfer(ADC1, LL_ADC_REG_DMA_TRANSFER_UNLIMITED);
a   LL_ADC_REG_SetSequencerLength(ADC1, LL_ADC_REG_SEQ_SCAN_DISABLE);
8 b LL_ADC_REG_SetSequencerRanks(ADC1, LL_ADC_REG_RANK_1, LL_ADC_CHANNEL_6);
  c LL_ADC_SetChannelSamplingTime(ADC1, LL_ADC_CHANNEL_6, LL_ADC_SAMPLINGTIME_12CYCLES_5);
9   LL_ADC_DisableDeepPowerDown(ADC1);
    LL_ADC_EnableInternalRegulator(ADC1);
10  //wait 20us for internal regulator stabilization
```

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project - DAC_DMA configuration

The task is to configure Channel 3 in DMA1 to work with channel 1 in DAC1 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)*.

Before configuration we should connect clock to the DMA1 peripheral.
To do this we can use dedicated macro: **__HAL_RCC_PPP_CLK_ENABLE()**

After 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 unitary LL

3 - LL_L4_DAC_ADC project - DAC_DMA configuration

The task is to configure Channel 3 in DMA1 to work with DAC1, channel1 in continuous (circular) mode.

1. First step should be connection of the bus clock to DMA1 peripheral.
   We can use previously used macro or dedicated function **LL_AHB1_GRP1_EnableClock()**
2. Further we need to configure DMA1, channel3 (using **LL_DMA_ConfigTransfer()** function) in the following way:
   a. DMA transfer in circular mode to match with DAC1 configuration: DMA unlimited requests.
   b. DMA transfer from memory with address increment.
   c. DMA transfer to DAC1 without address increment by half-word to match with DAC1 configuration: DAC1 resolution 12 bits.
   d. DMA transfer from memory by half-word to match with DAC1 conversion data buffer variable type: half-word.
3. Further we should assign channel3 of DMA1 to DAC1 request (point 10.4.7, table 39 in reference manual) using **LL_DMA_SetPeriphRequest()** function.

```
/* DAC DMA LL configuration */
```

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project - DAC_DMA configuration

The task is to configure Channel 3 in DMA1 to work with DAC1, channel1 in continuous (circular) mode.

1. First step should be connection of the bus clock to DMA1 peripheral.
   We can use previously used macro or dedicated function **LL_AHB1_GRP1_EnableClock()**
2. Further we need to configure DMA1, channel3 (using **LL_DMA_ConfigTransfer()** function) in the following way:
   a. DMA transfer in circular mode to match with DAC1 configuration: DMA unlimited requests.
   b. DMA transfer from memory with address increment.
   c. DMA transfer to DAC1 without address increment by half-word to match with DAC1 configuration: DAC1 resolution 12 bits.
   d. DMA transfer from memory by half-word to match with DAC1 conversion data buffer variable type: half-word.
3. Further we should assign channel3 of DMA1 to DAC1 request (point 10.4.7, table 39 in reference manual) using **LL_DMA_SetPeriphRequest()** function.

```
/* DAC DMA LL configuration */
1 LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_DMA1); //enable clock
2 LL_DMA_ConfigTransfer(DMA1,
                        LL_DMA_CHANNEL_3,
                        LL_DMA_DIRECTION_MEMORY_TO_PERIPH |
                        LL_DMA_MODE_CIRCULAR              |
                        LL_DMA_PERIPH_NOINCREMENT         |
                        LL_DMA_MEMORY_INCREMENT           |
                        LL_DMA_PDATAALIGN_HALFWORD        |
                        LL_DMA_MDATAALIGN_HALFWORD        |
                        LL_DMA_PRIORITY_HIGH               );
3 LL_DMA_SetPeriphRequest(DMA1, LL_DMA_CHANNEL_3, LL_DMA_REQUEST_6);
```

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project - DAC_DMA configuration

The task is to configure Channel 3 in DMA1 to work with DAC1, channel1 in continuous (circular) mode.

4. In the next step we should configure DMA transfer addresses of source (dacbuf[] buffer) and destination (DAC1 data register for 12bit data aligned to right) using **LL_DMA_ConfigAddresses()** function *).
5. Further we need to configure DMA transfer size (size of the dacbuf[]) using **LL_DMA_SetDataLength()** function
6. At the end we should enable 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 unitary LL

3 - LL_L4_DAC_ADC project - DAC_DMA configuration

The task is to configure Channel 3 in DMA1 to work with DAC1, channel1 in continuous (circular) mode.

4. In the next step we should configure DMA transfer addresses of source (dacbuf[] buffer) and destination (DAC1 data register for 12bit data aligned to right) using **LL_DMA_ConfigAddresses()** function *).
5. Further we need to configure DMA transfer size (size of the dacbuf[]) using **LL_DMA_SetDataLength()** function
6. At the end we should enable channel3 in DMA1 using **LL_DMA_EnableChannel()** function

```
4  LL_DMA_ConfigAddresses(DMA1,
         LL_DMA_CHANNEL_3,
         (uint32_t)&dacbuf,
         LL_DAC_DMA_GetRegAddr(DAC1,LL_DAC_CHANNEL_1,LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED),
         LL_DMA_DIRECTION_MEMORY_TO_PERIPH);

5  LL_DMA_SetDataLength(DMA1, LL_DMA_CHANNEL_3, DACBUFSIZE);
   LL_DMA_EnableChannel(DMA1, LL_DMA_CHANNEL_3);
6
```

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

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project - ADC_DMA configuration

The task is to configure Channel 1 in DMA1 to work with channel 6 in ADC1 in the following way:

- Continuously read data (in halfwords, without pointer incrementation) from ADC1 data register for regular channels (12bits)*.
- Continuously write data to internal buffer (adcbuf[ADCBUFSIZE]) in halfwords, with pointer incrementation

Before the configuration we should connect the clock to the DMA1 peripheral. To do this we can use there dedicated macro: **__HAL_RCC_PPP_CLK_ENABLE()**

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 unitary LL

3 - LL_L4_DAC_ADC project - ADC_DMA configuration

The task is to configure Channel 1 in DMA1 to work with ADC1, channel6 in continuous (circular) mode.

1. First step should be connection of the bus clock to DMA1 peripheral. We can use previously used macro or dedicated function **LL_AHB1_GRP1_EnableClock()**
2. Further we need to configure DMA1, channel1 (using **LL_DMA_ConfigTransfer()** function) in the following way:
   a. DMA transfer in circular mode to match with ADC1 configuration: DMA unlimited requests.
   b. DMA transfer from ADC1 data register for regular conversions without address increment by half-word to match with ADC1 configuration: ADC1 resolution 12 bits.
   c. DMA transfer to memory with address increment.
   d. DMA transfer to memory by half-word to match with ADC1 conversion data buffer variable type: half-word.
3. Further we should assign channel1 of DMA1 to ADC1 request (point 10.4.7, table 39 in reference manual) using **LL_DMA_SetPeriphRequest()** function.

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

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project - ADC_DMA configuration

The task is to configure Channel 1 in DMA1 to work with ADC1, channel6 in continuous (circular) mode.

1. First step should be connection of the bus clock to DMA1 peripheral. We can use previously used macro or dedicated function **LL_AHB1_GRP1_EnableClock()**
2. Further we need to configure DMA1, channel1 (using **LL_DMA_ConfigTransfer()** function) in the following way:
   a. DMA transfer in circular mode to match with ADC1 configuration: DMA unlimited requests.
   b. DMA transfer from ADC1 data register for regular conversions without address increment by half-word to match with ADC1 configuration: ADC1 resolution 12 bits.
   c. DMA transfer to memory with address increment.
   d. DMA transfer to memory by half-word to match with ADC1 conversion data buffer variable type: half-word.
3. Further we should assign channel1 of DMA1 to ADC1 request (point 10.4.7, table 39 in reference manual) using **LL_DMA_SetPeriphRequest()** function.

```
/* ADC DMA LL configuration */
1 LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_DMA1); //enable clock
2 LL_DMA_ConfigTransfer(DMA1,
                        LL_DMA_CHANNEL_1,
                        LL_DMA_DIRECTION_PERIPH_TO_MEMORY |
                        LL_DMA_MODE_CIRCULAR              |
                        LL_DMA_PERIPH_NOINCREMENT         |
                        LL_DMA_MEMORY_INCREMENT           |
                        LL_DMA_PDATAALIGN_HALFWORD        |
                        LL_DMA_MDATAALIGN_HALFWORD        |
                        LL_DMA_PRIORITY_HIGH               );
3 LL_DMA_SetPeriphRequest(DMA1, LL_DMA_CHANNEL_1, LL_DMA_REQUEST_0);
```

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project - ADC_DMA configuration

The task is to configure Channel 1 in DMA1 to work with ADC1, channel6 in continuous (circular) mode.

4. In the next step we should configure DMA transfer addresses of source (ADC1 data register for 12bit data) and destination (adcbuf[] buffer) using **LL_DMA_ConfigAddresses()** function *).
5. Further we need to configure DMA transfer size (size of the adcbuf[]) using **LL_DMA_SetDataLength()** function
6. At the end we should enable 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 unitary LL

3 - LL_L4_DAC_ADC project - ADC_DMA configuration

The task is to configure Channel 1 in DMA1 to work with ADC1, channel6 in continuous (circular) mode.

4. In the next step we should configure DMA transfer addresses of source (ADC1 data register for 12bit data) and destination (adcbuf[] buffer) using **LL_DMA_ConfigAddresses()** function *).
5. Further we need to configure DMA transfer size (size of the adcbuf[]) using **LL_DMA_SetDataLength()** function
6. At the end we should enable channel1 in DMA1 using **LL_DMA_EnableChannel()** function

4
```
LL_DMA_ConfigAddresses(DMA1,
        LL_DMA_CHANNEL_1,
        LL_ADC_DMA_GetRegAddr(ADC1, LL_ADC_DMA_REG_REGULAR_DATA),
        (uint32_t)&adcbuf,
        LL_DMA_DIRECTION_PERIPH_TO_MEMORY);
```
5
```
LL_DMA_SetDataLength(DMA1, LL_DMA_CHANNEL_1, ADCBUFSIZE);
```
```
LL_DMA_EnableChannel(DMA1, LL_DMA_CHANNEL_1);
```
6

life.augmented

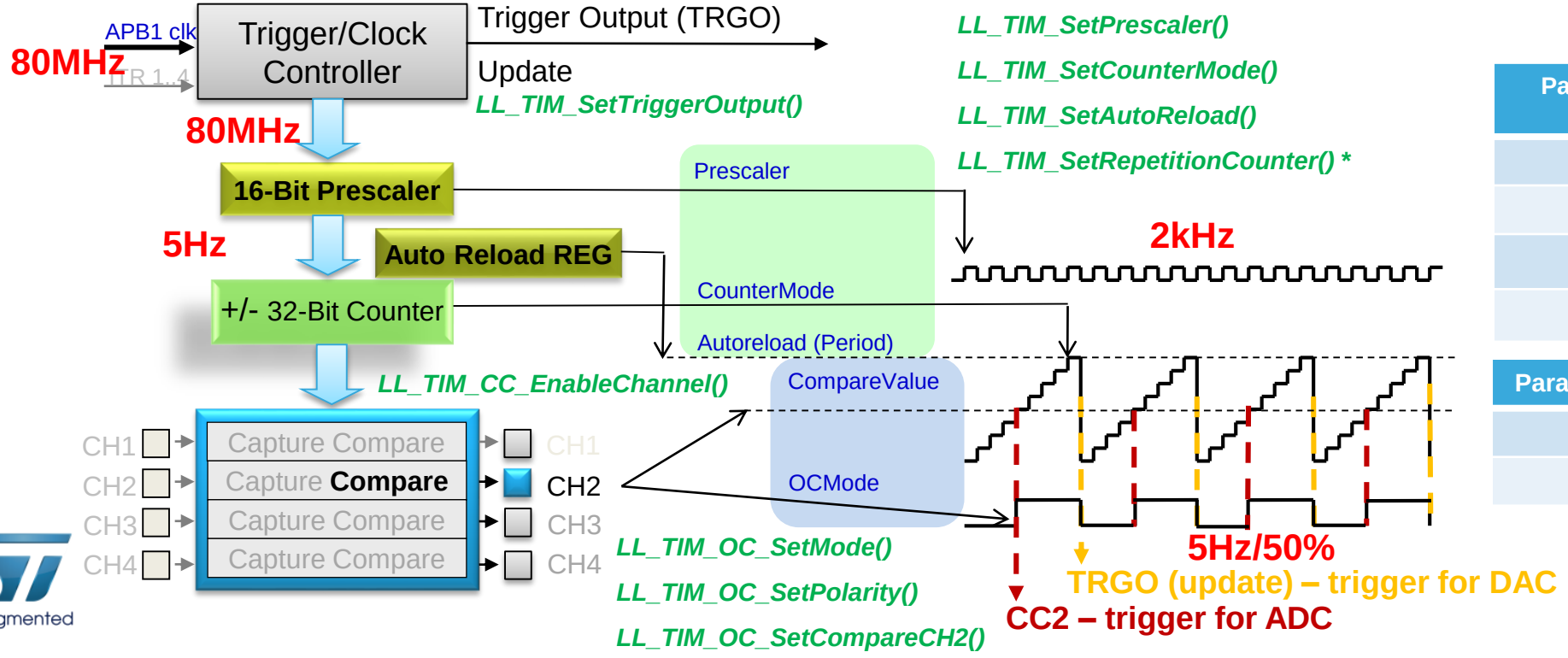
*) Hint: To get the proper address we can use function **LL_ADC_DMA_GetRegAddr()**

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project – TIM2 configuration

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

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



Parameter for timer2 timebase	value
ClockDivision	?
Prescaler	?
CounterMode	?
Autoreload	?

Parameter for channel 2	value
CompareValue	?
OCMode	?

life.augmented

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project – TIM2 timebase configuration

The task is to configure Timer2 to work in OC toggle mode on channel2 (f=5Hz, 50%) without connection to the pin and with TRGO signal configured to update event.

1. First step should be connecting clock to Timer2 (TIM2) using macro **__HAL_RCC_TIM2_CLK_ENABLE()** or dedicated function **LL_APB1_GRP1_EnableClock()**
2. Further we should set prescaler for timer2 using **LL_TIM_SetPrescaler()** function
3. Further we should configure autoreload (period value) using **LL_TIM_SetAutoReload()** function
4. Further we should configure counter mode to up-counting using **LL_TIM_SetCounterMode()** function
5. Further we should disable repetition counter by writing 0 to this counter using **LL_TIM_SetRepetitionCounter()** function

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project – TIM2 timebase configuration

The task is to configure Timer2 to work in OC toggle mode on channel2 (f=5Hz, 50%) without connection to the pin and with TRGO signal configured to update event.

1. First step should be connecting clock to Timer2 (TIM2) using macro **__HAL_RCC_TIM2_CLK_ENABLE()** or dedicated function **LL_APB1_GRP1_EnableClock()**
2. Further we should set prescaler for timer2 using **LL_TIM_SetPrescaler()** function
3. Further we should configure autoreload (period value) using **LL_TIM_SetAutoReload()** function
4. Further we should configure counter mode to up-counting using **LL_TIM_SetCounterMode()** function
5. Further we should disable repetition counter by writing 0 to this counter using **LL_TIM_SetRepetitionCounter()** function

```
/* TIM2 LL configuration */
1 LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_TIM2);
2 LL_TIM_SetPrescaler(TIM2, 39999);
3 LL_TIM_SetAutoReload(TIM2, 399);
4 LL_TIM_SetCounterMode(TIM2, LL_TIM_COUNTERMODE_UP);
5 LL_TIM_SetRepetitionCounter(TIM2, 0);
```

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project – TIM2 OC2 configuration

The task is to configure Timer2 to work in OC toggle mode on channel2 (f=5Hz, 50%) without connection to the pin and with TRGO signal configured to update event.

1. Our next step should be configuration of trigger output (TRGO) to be connected to update event.
   We can use **LL_TIM_SetTriggerOutput()** function
2. Further we should configure channel2 into output compare mode using the following procedure:
   a. Set output compare mode to TOGGLE using **LL_TIM_OC_SetMode()** function
   b. Set output channel polarity to OC active high using **LL_TIM_OC_SetPolarity()** function
   c. Set pulse value using **LL_TIM_OC_SetCompareCH2()** function
3. Further we should enable channel2 in timer2 using **LL_TIM_CC_EnableChannel()** function

Replacing HAL functions with unitary LL

3 - LL_L4_DAC_ADC project – TIM2 OC2 configuration

The task is to configure Timer2 to work in OC toggle mode on channel2 (f=5Hz, 50%) without connection to the pin and with TRGO signal configured to update event.

1. Our next step should be configuration of trigger output (TRGO) to be connected to update event.
   We can use **LL_TIM_SetTriggerOutput()** function
2. Further we should configure channel2 into output compare mode using the following procedure:
   a. Set output compare mode to TOGGLE using **LL_TIM_OC_SetMode()** function
   b. Set output channel polarity to OC active high using **LL_TIM_OC_SetPolarity()** function
   c. Set pulse value using **LL_TIM_OC_SetCompareCH2()** function
3. Further we should enable channel2 in timer2 using **LL_TIM_CC_EnableChannel()** function

```
1  LL_TIM_SetTriggerOutput(TIM2, LL_TIM_TRGO_UPDATE);
a  LL_TIM_OC_SetMode(TIM2, LL_TIM_CHANNEL_CH2, LL_TIM_OCMODE_TOGGLE);
2b LL_TIM_OC_SetPolarity(TIM2, LL_TIM_CHANNEL_CH2, LL_TIM_OCPOLARITY_HIGH);
c  LL_TIM_OC_SetCompareCH2(TIM2, 200);
3  LL_TIM_CC_EnableChannel(TIM2, LL_TIM_CHANNEL_CH2);
```

Replacing HAL functions with unitary LL

4 - LL_L4_DAC_ADC 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 unitary LL

4 - LL_L4_DAC_ADC 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);
```

HAL vs. LL libraries

offer		portability	Optimization (memory & MIPS)	easy	readiness	Hardware coverage
STM32Cube	HAL API	+++	+	++	+++	+++
	LL APIs	+	+++	+	++	++

What have we learnt?

- ✓Gain knowledge about complete ST software offer for STM32 microcontrollers
- ✓Gain knowledge about Low Layer Library concepts: unitary and init
- ✓Practice Low Layer Library concept on previously generated HAL based project
- ✓Gain knowledge about differences between HAL and LL concepts.

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