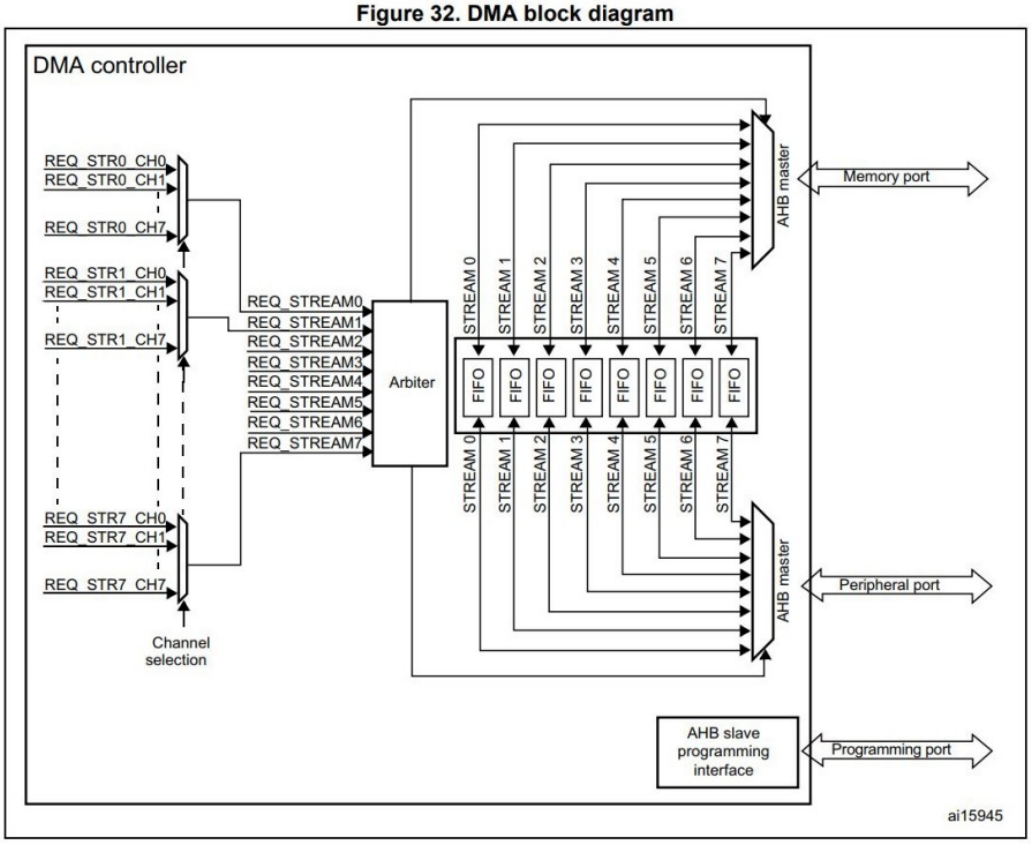
Lab 7 - DMA

In this lab, you will be using DMA (Direct-memory-access) to transfer data. DMA allows your application to continue working while data can be copied from one location to another. STM32 has two DMA controllers. Each DMA has the following block diagram.

Figure 32. DMA block diagram



Each controller is connected to different bus. Please see additional information in the reference manual. Each controller can work in this following operations:

- Memory to Memory
- Peripheral to Memory
- Memory to Peripheral

When the controller is working in peripheral to memory or memory to peripheral, the controller will transfer data upon "request" input. The request table can be found in Table 42. of STM32F4 Reference Manual.

Your Tasks:

1. The first task is to use memory to memory feature of the DMA. You program will do the following
   - Create two arrays of the 255 bytes, arr0 and arr1.
   - Fill the first array with arr0[i] = ((i + 55) * 37) % 57;
   - Fill the second array with zero
   - Upon use hitting the USER button, it will copy data from arr0 to arr1
   - Inspect data in arr1 using the debugger

   Because of the bus structure of STM32, you can only do memory to memory transfer using DMA2 controller.

2. In the second task, your program must do the following:

- Accept data through UART. However, it will return data only after 256 characters are received.

- During the time STM32 is sending the data back to the computer, it must still accept more input, and would send the data back after 256 characters.

- An LED must always brink 200ms period.

- You must use DMA for accepting data from UART, and use DMA interrupted at the end.

3. The last task is similar to the second one, however, it will send data back at the rate of 10 characters per second. (Optional, but if you have time, try it)

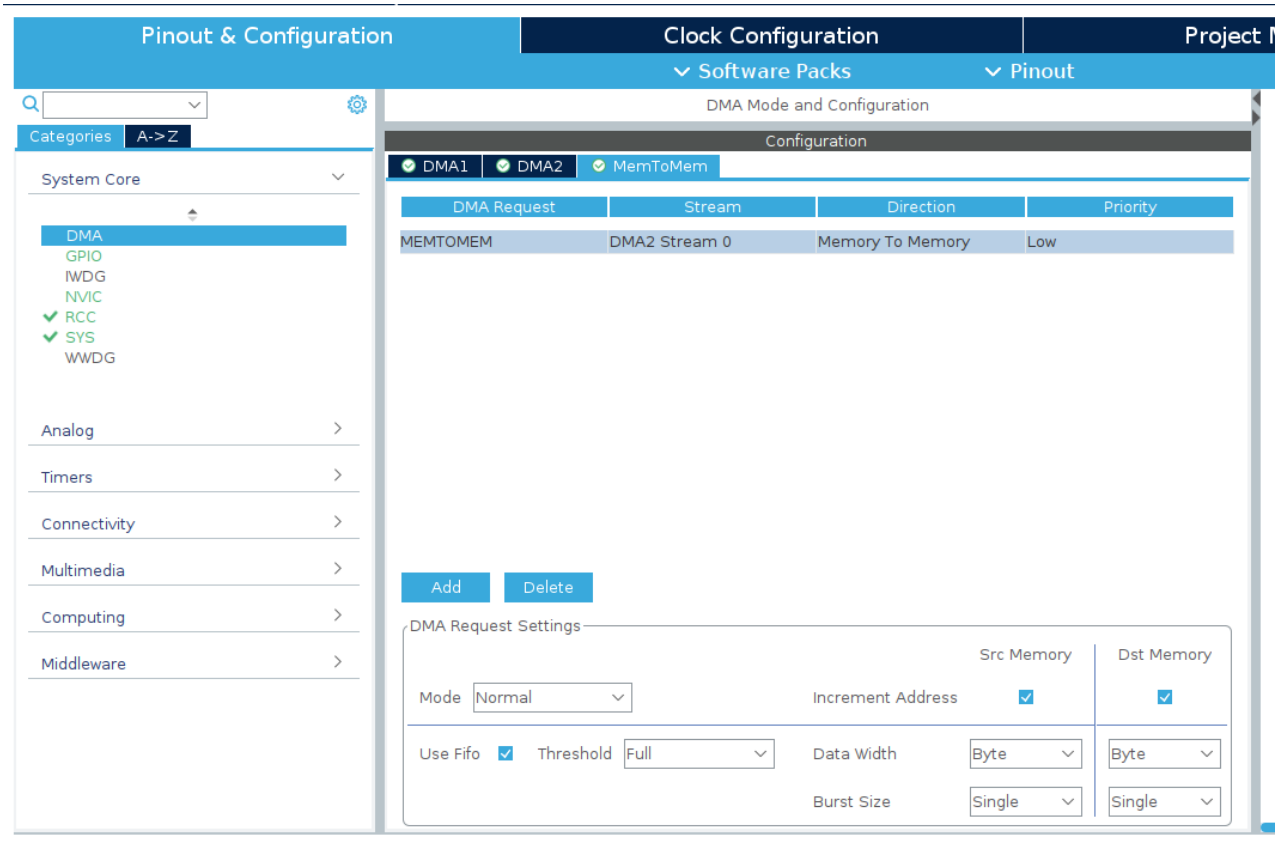
Remark

1. You must implement all of these memory/peripheral operations using DMA in order to complete the task.

2. For Task 3, since the rate of transfer is much slower than the rate of UART, you can use TIMER to trigger interrupt request. Note that you will be using a different DMA Stream/Channel than the typical UART one.

Hint:

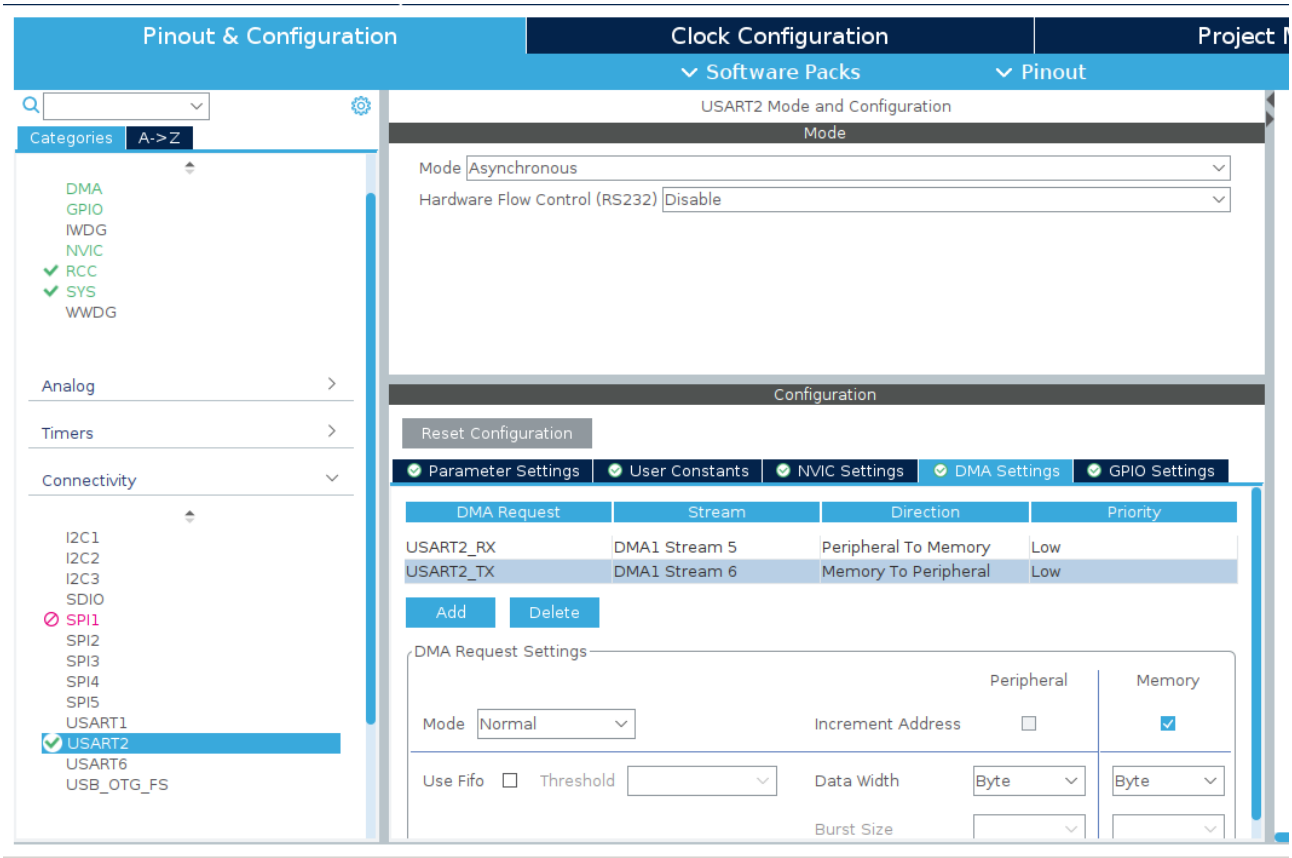
For Task 1. You can use STM32Cube to setup most of the DMA by going to DMA→ MemToMem → Add



Then you can start DMA transfer with

```
HAL_DMA_Start(&hdma_memtomem_dma2_stream0, arr0, arr1, 256);
```

For Task 2: You can add DMA in STM32CubeIDE by going to USART2 → DMA Settings → Add Then choose USART2_RX then add again and add USART2_TX



Then you can started transfer from UART to memory (RX) using

```
HAL_UART_Receive_DMA(&huart2, buffer, 256);
```

or memory to UART (TX) using

```
HAL_UART_Transmit_DMA(&huart2, buffer, 256);
```

When the transmitted is completed, it will call the following functions (you have to declare these functions in your program)

```
HAL_UART_TxHalfCpltCallback()  // Half way transmit
HAL_UART_TxCpltCallback()      // Finish transmit
HAL_UART_RxHalfCpltCallback()  // Half way receive
HAL_UART_RxCpltCallback()      // Finish received
```

There are two easy ways to implement the task. The first method use ping-pong buffers, i.e. when you read from one buffer and finish, you can start transfer for the next one using another buffer, then switch back. Something like this, (the code is not tested.)

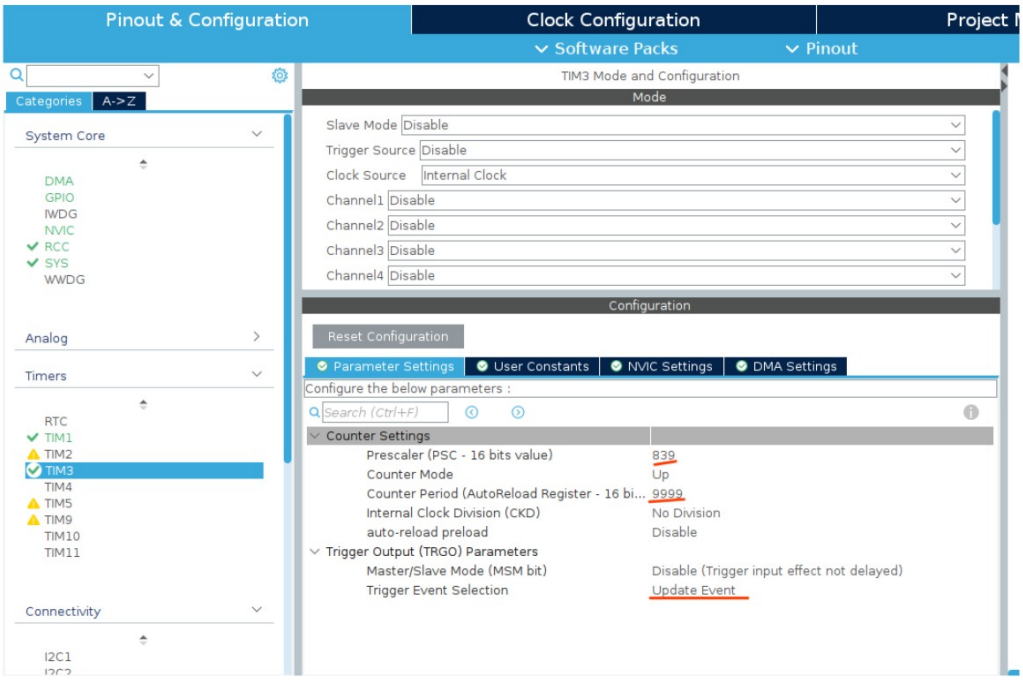
```
int current_buffer_idx = 0;
char buffer[2][256];


void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
      HAL_UART_Transmit_DMA(&huart2, buffer[current_buffer_idx], 256);
      current_buffer_idx = !current_buffer_idx;
      HAL_UART_Receive_DMA(&huart2, buffer[current_buffer_idx], 256);
}
```

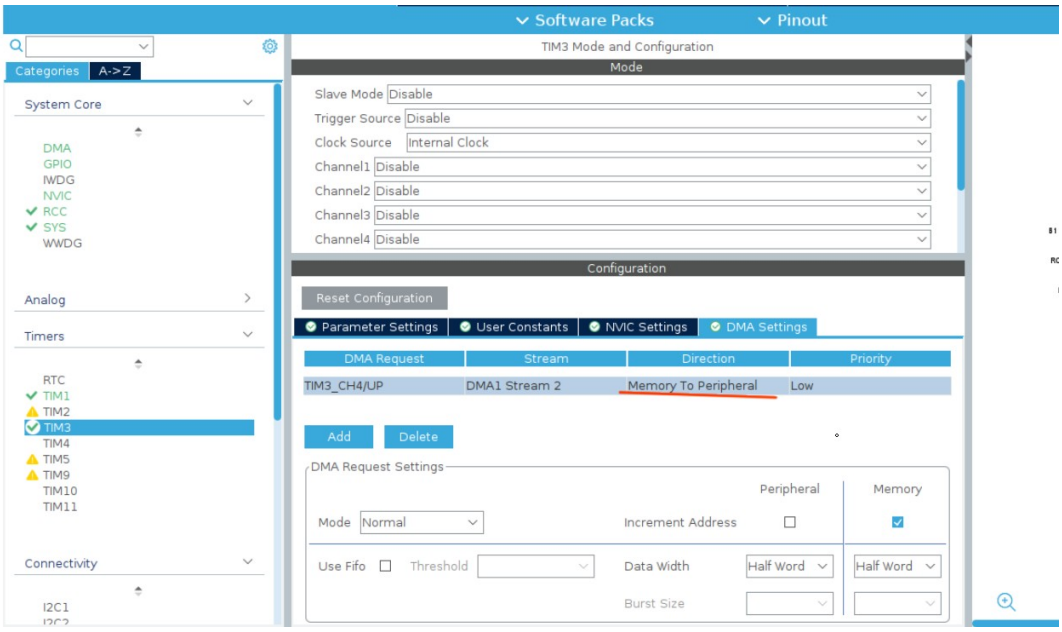
```
// In size main function,
HAL_UART_Receive_DMA(&huart2, buffer[current_buffer_idx], 256);
```

Another method is to set the buffer to mode to be circular and transfer size 512, and use half complete instead. It's a bit more complicated……

For Task 3: This is a bit more complicated. There are several methods to do it. But in general, you have to setup timer so that it is sending out event at 10Hz. Something like this



Then create a DMA,



In the main function, you can start timer with

```
HAL_TIM_Base_Start(&htim3);
__HAL_TIM_ENABLE_DMA(&htim3, TIM_DMA_UPDATE);
```

Note that we enable DMA update signal.

Then when you want to transfer the data, you can do the following

```
HAL_DMA_Start_IT(&hdma_tim3_ch4_up, buffer[current_buffer_idx], huart2.Instance->DR, 256);
```

Note that the code is **untested. It is a hint. Have some adventure!!.**