



life.augmented



STM32H5 Workshop GPDMA

| | | | | | |
|--------------|---------|------|------------|----------|------------|
| Temp Min | 15.56 C | Date | 05-03-2023 | Customer | World Wide |
| Temp Max | 31.94 C | Time | 12:30 | Status | Working |
| Temp Ambient | 22.13 C | Job | W0725014 | Scale | None |

Agenda

#

Overview



#

Structure of example



#

ADC+UART+TIM
with LLI controlled GPDMA



#

Conclusion



#

What's next



Theory



Hands-on

DMA overview

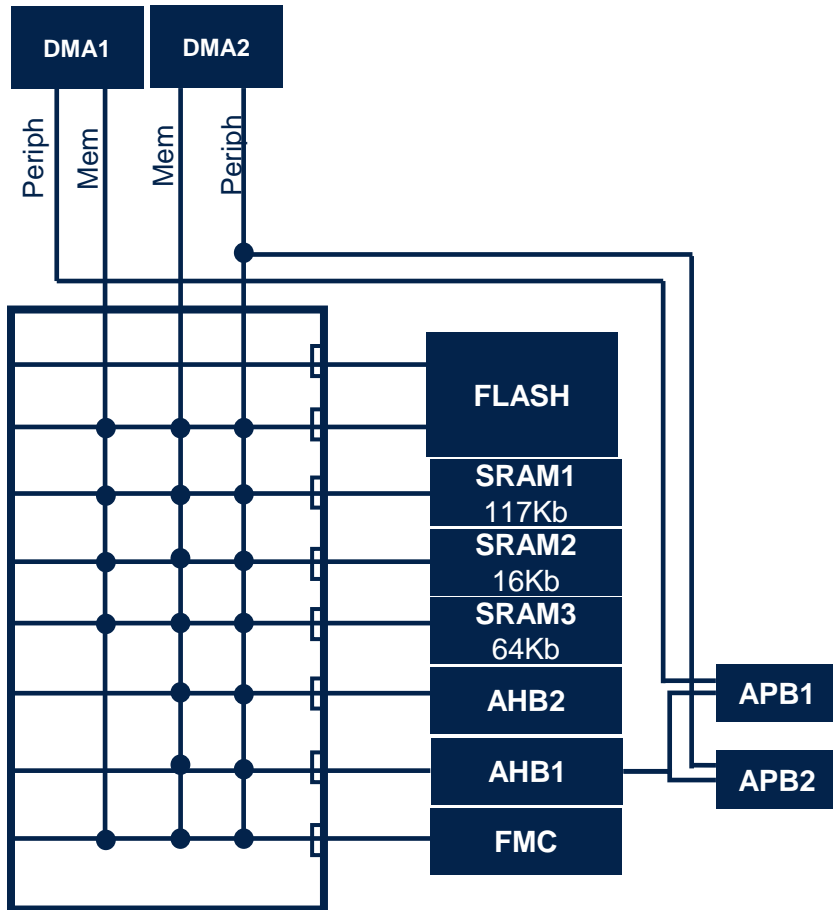
DMA overview

- A new DMA module
 - 2 hardware instances
 - GPDMA with symmetric configuration
 - Dual port DMA with dedicated path to APB
 - Integrated DMAMUX features
 - Linked-list based programming
 - Flexible intra-channel and inter-channel input/output control
 - Run-time isolation features

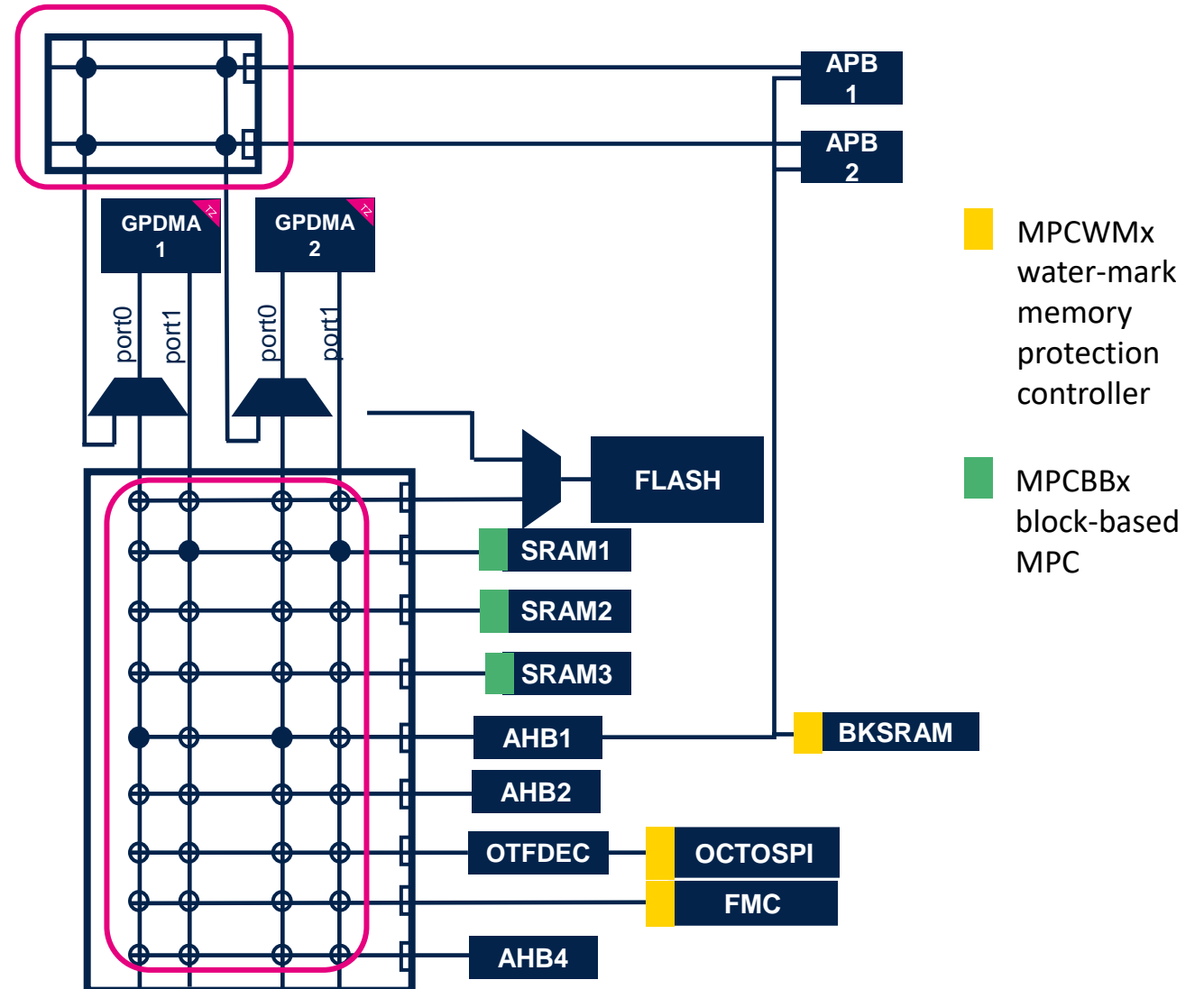
Application benefit

- Off-load CPU for data transfers from a memory-mapped source to a memory-mapped destination

STM32F4x9



STM32H563 GPDMA integration



GPDMA key features 1/2

- Bidirectional AHB master port(s): GPDMA1/2: 2 ports
- Memory-mapped data transfers from a source to a destination
 - Peripheral-to-memory
 - Memory-to-peripheral
 - Memory-to-memory
 - Peripheral-to-peripheral
- Autonomous data transfers during Sleep mode
- 8 Concurrent DMA channels for each controller
- Transfers arbitration is based on a 4-grade priority policy
 - One reserved highest priority queue for time-sensitive traffic
 - Three lower priority queues with weighted round robin allocation

GPDMA key features 2/2

- **GPDMA data handling**

- Byte-based reordering
- padding/truncation
- left / right alignment
- packing/unpacking
- sign extension

- **Linear addressing mode**

- Fixed addressing (typically for peripheral data register)
- Contiguously-incremented addressing (typically for memory access)
- Blocks up to 64kB (16-bit BNDT)

- **2D addressing mode (ch6..7), additional**

- Repeated block mode: programmable repeated block counter (11-bit BRC, up to 2k blocks)
- Programmable source/destination signed burst address offset (2x 14bit, up to +/-8kB)
 - Non-contiguous incremented/decremented addressing after each burst
- Programmable source/destination signed block address offset (2x 17bit, up to +/-64kB)
 - Non-contiguous incremented/decremented addressing after each block

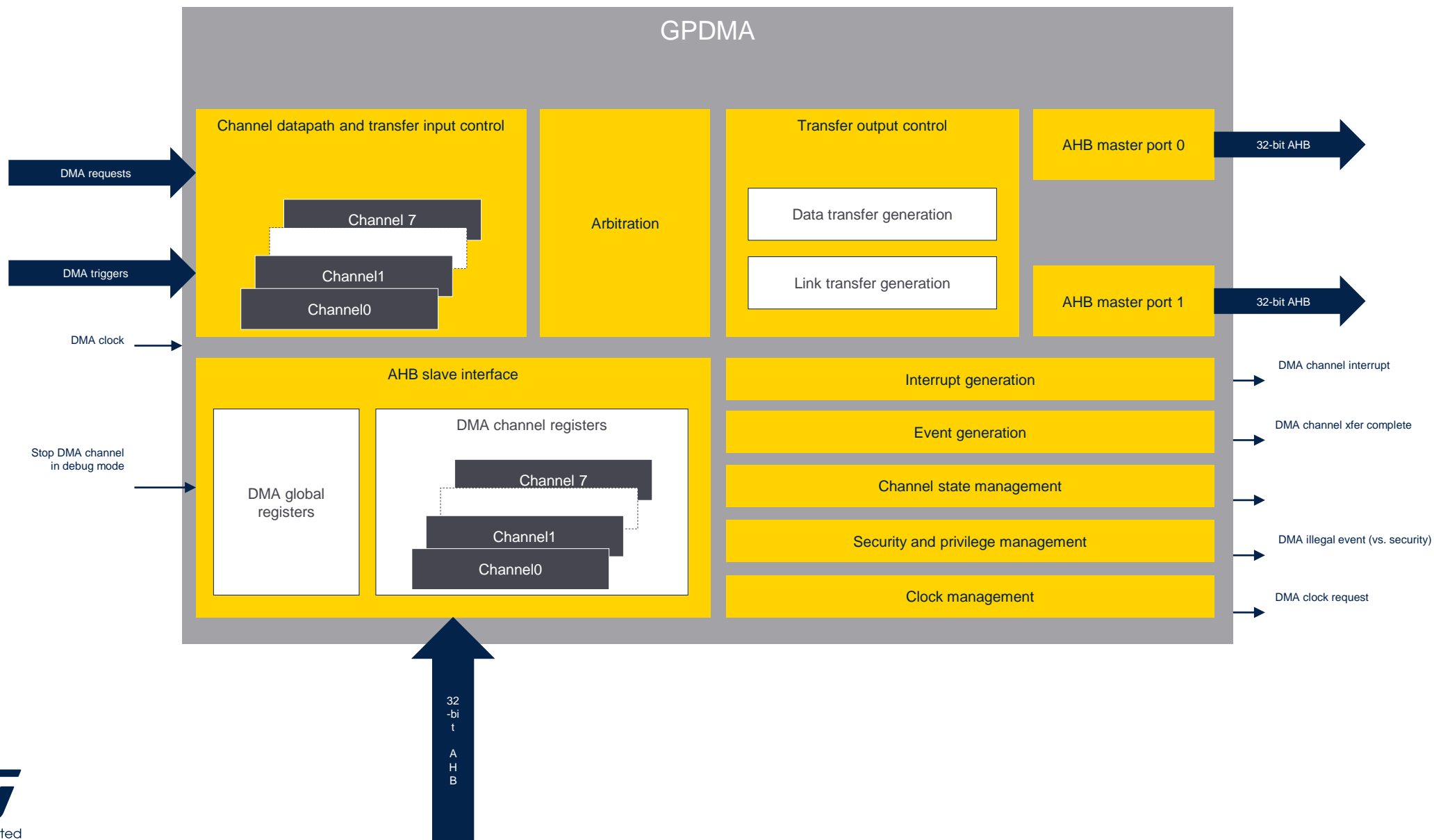
GPDMA linked-list register file

The GPDMA will use nodes for those registers

- TR1 - Transfer register 1
- TR2 - Transfer register 2
- BR1 - Block register 1
- SAR - Source address register
- DAR - Destination address register
- TR3 - Transfer register 3 ... SA & DA offsets increment for **2D**
- BR2 - Block register 2 ... block repeated SA & DA offsets for **2D**
- LLR - Linker list register ... link to next LL node & update parameters

In our case each linked list node will update this GPDMA registers after previous GPDMA node is finished. This is automatically reconfiguring the GPDMA channel.

DMA block diagram



DMA specific implementation & user guidelines

| Feature | GPDMA |
|---|---|
| Master port(s) | 2x (32-bit) AHB (*) |
| DMA transfers | Single and bursts |
| DMA scheduler | FIFO-based bursts (dual issue) |
| Number of channels | 8 |
| Channel FIFO size | Ch0-3 (**): 8 bytes (2 words) Ch4-7 (**): 32 bytes (8 words) |
| Channel addressing mode | Ch0-5: linear Ch6-7: 2D addressing |
| Channel with peripheral early termination (I3C) | Ch0 and Ch7 |
| Maximum request ID | 135 |
| Maximum trigger ID | 43 |

(*): GPDMA master ports:

- Port#0 should be typically allocated for transfers to/from peripherals
 - There is a direct hardware datapath to APBx peripherals, out of the AHB matrix
 - AHB1 peripherals are the default slave target of the AHB matrix
- Port#1 should be typically allocated for transfers to/from memory
 - SRAM1 is the default slave
- In any case, any GPDMA target can be addressed from any port

(**): These channels should be typically allocated for transfers from/to an APB/AHB peripheral and SRAM

(***): These channels may be also used for transfers from/to a data-demanding AHB peripheral and SRAM, or for transfers from/to external memories

- 4-word burst should be privileged when applicable for faster performances (faster back-to-back transfers, lower bus utilization)

Structure of example

ADC and UART with LLI controlled GPDMA transfer triggered by TIMER

- Use **NUCLEO-H563ZI board** and **STM32CubeIDE**
- Setup ADC1 to
 - convert 4 channels in circle - channels 0, 2, 6, 13
 - generate DMA requests
- Setup USART3 to
 - transmit obtained data
- Setup TIM15 to
 - generate a 1 second trigger
- Set GPDMA with linked list
 - to get data from ADC and transfer them to buffer after trigger
 - to get data from buffer and transfer them to UART3

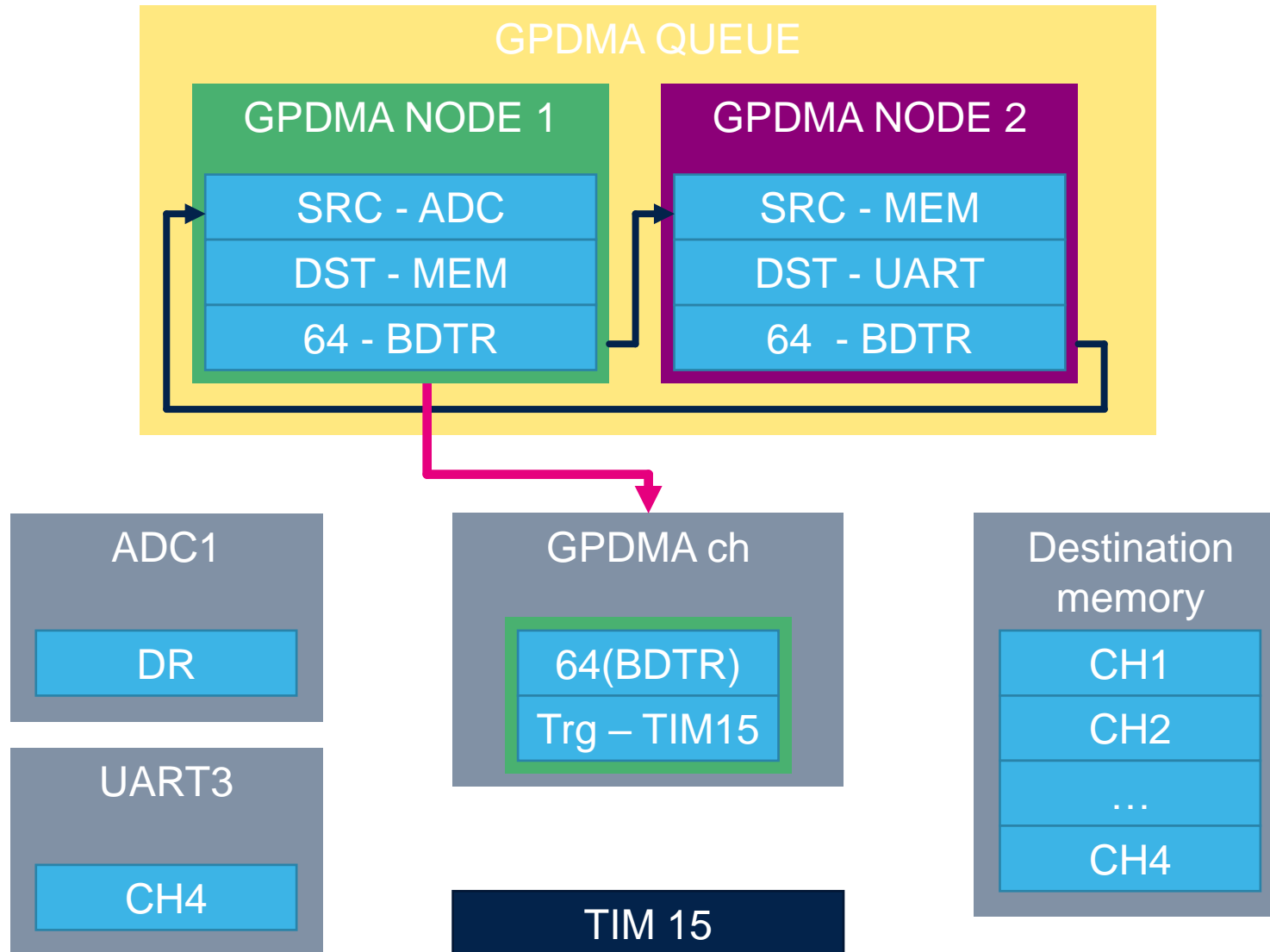


ADC + UART + TIM with LLI controlled GPDMA transfer

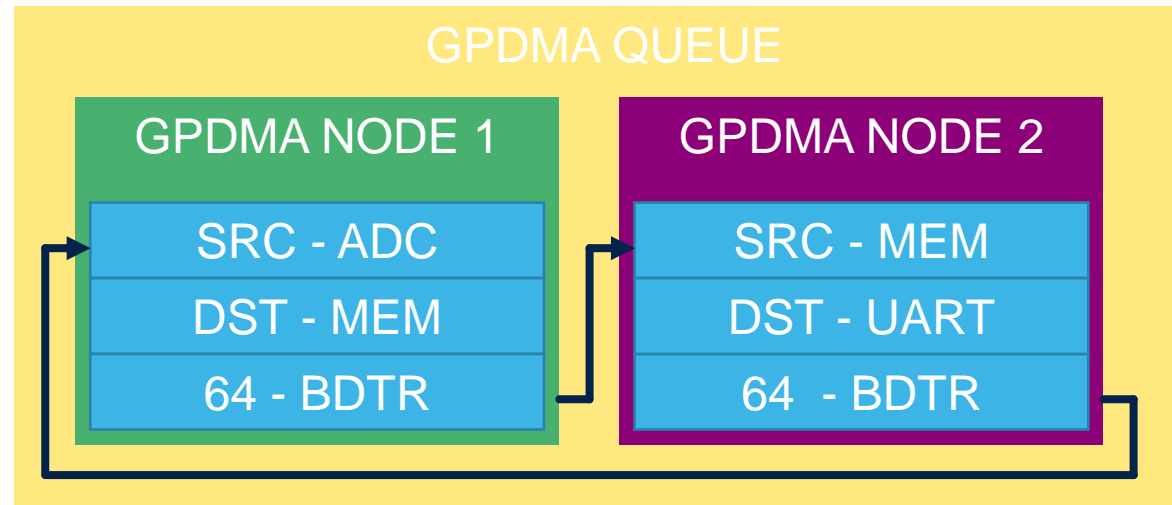
- If ADC and UART work in loop without a trigger, it may cause a crash of terminal due to high baud rate.
- We can slow down the GPDMA by adding trigger.
- As trigger source in our case, we will use a timer TIM15 with period of 1s.
- Then the GPDMA transfer will be conditioned by this trigger event.



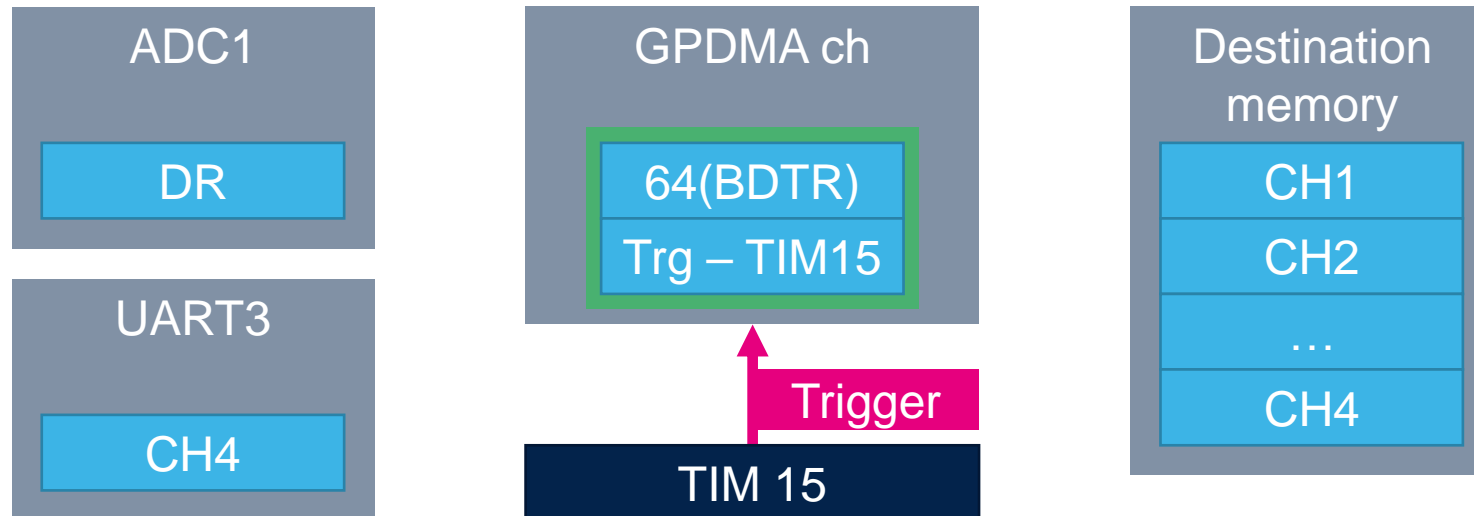
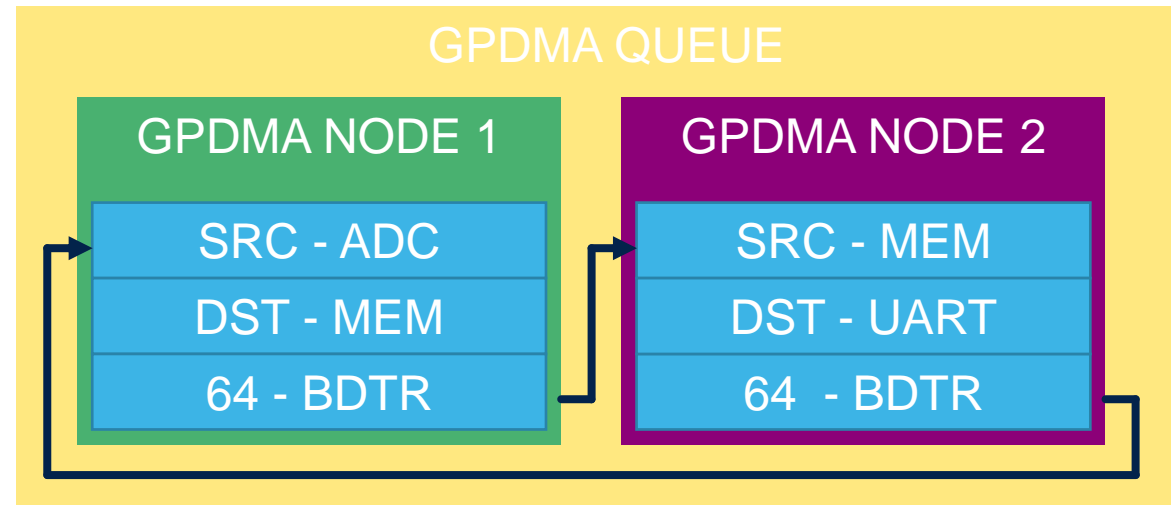
ADC + UART + TIM with LLI controlled GPDMA transfer



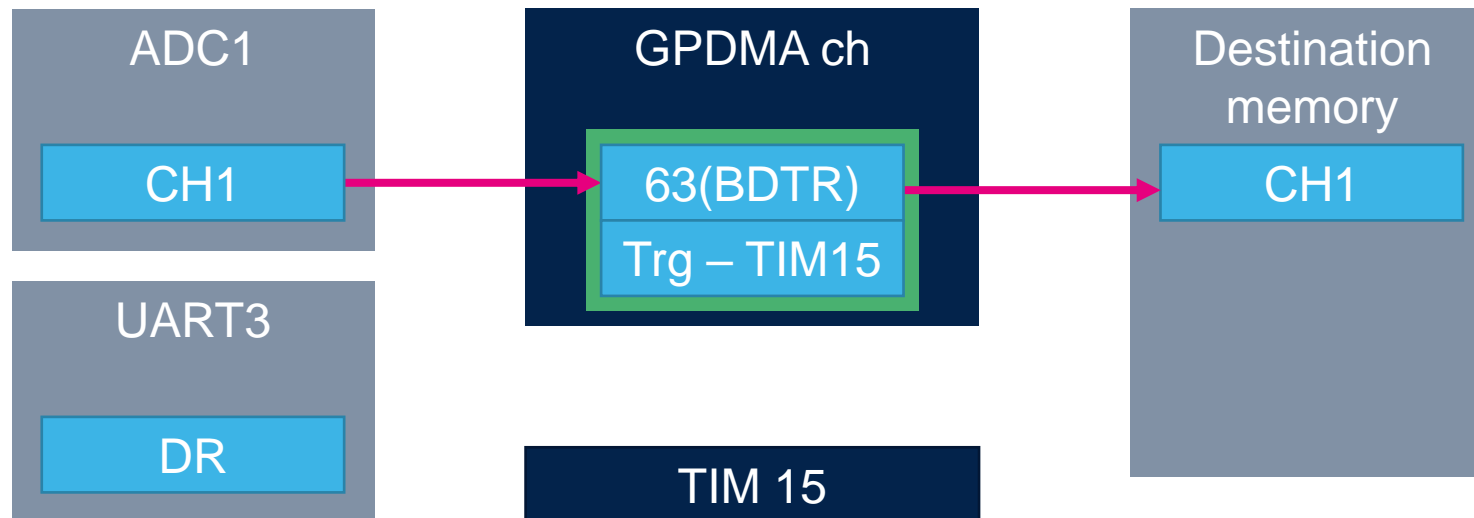
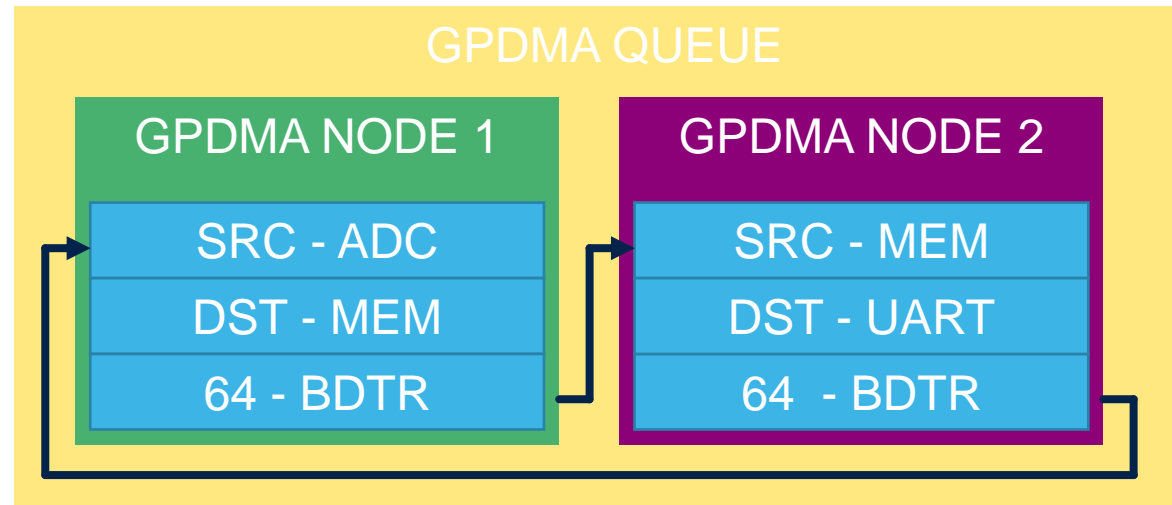
ADC + UART + TIM with LLI controlled GPDMA transfer



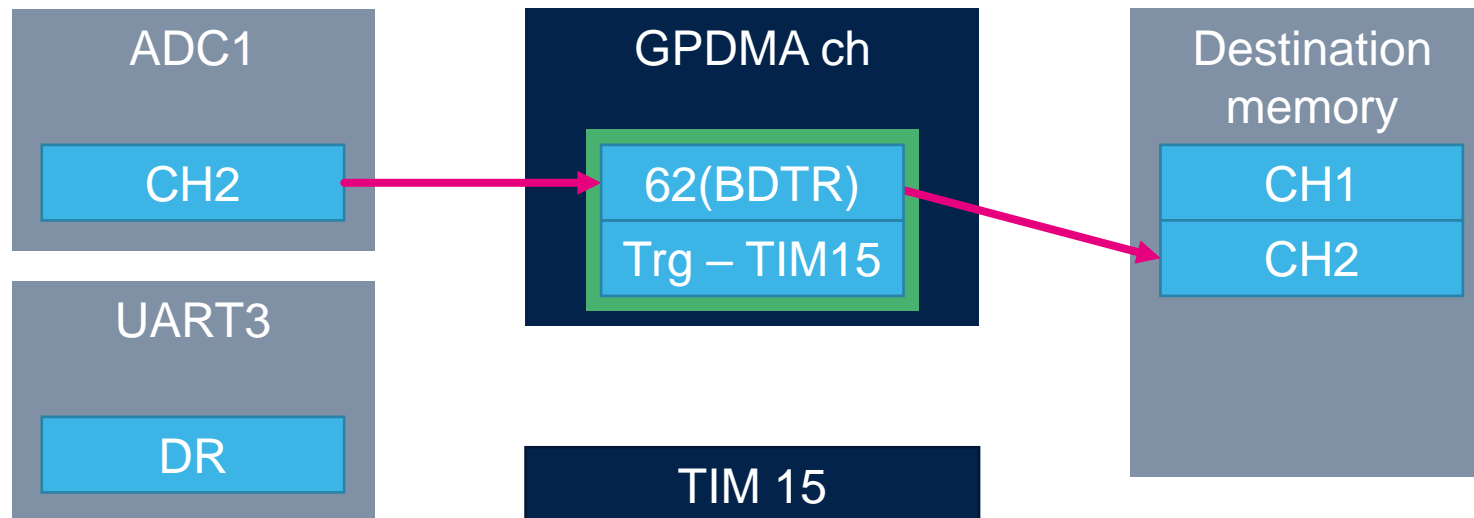
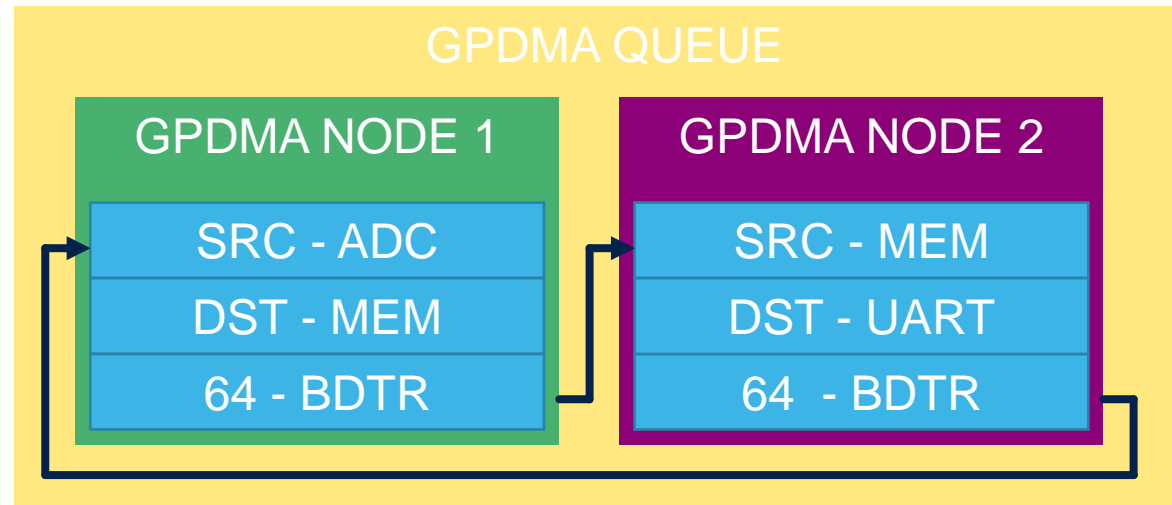
ADC + UART + TIM with LLI controlled GPDMA transfer



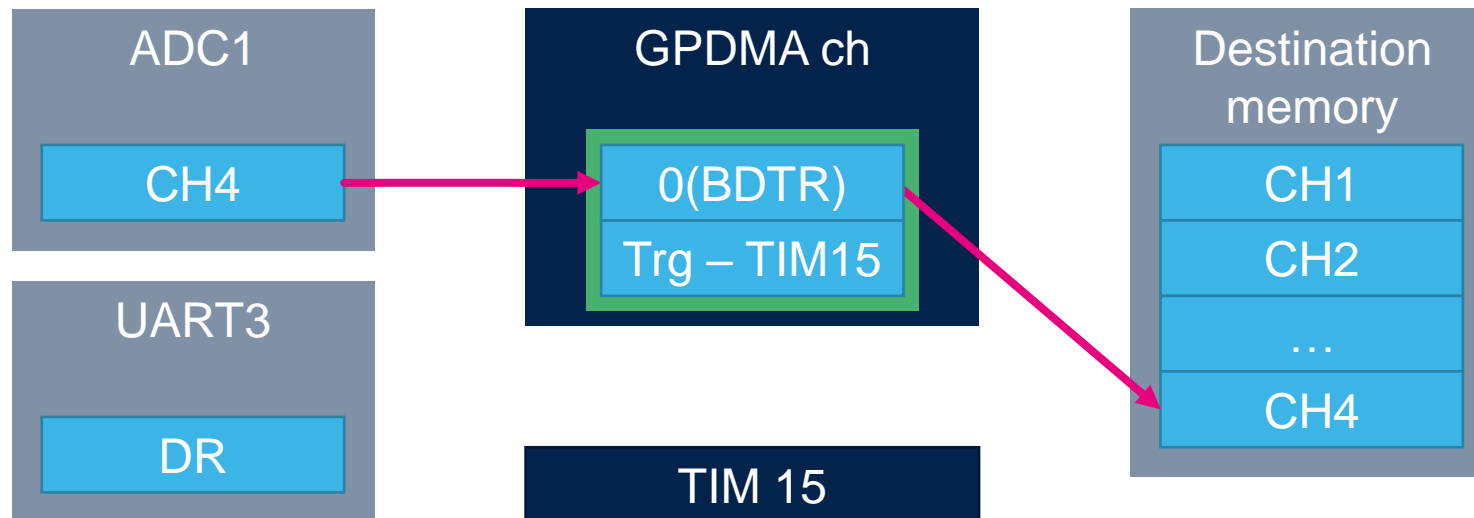
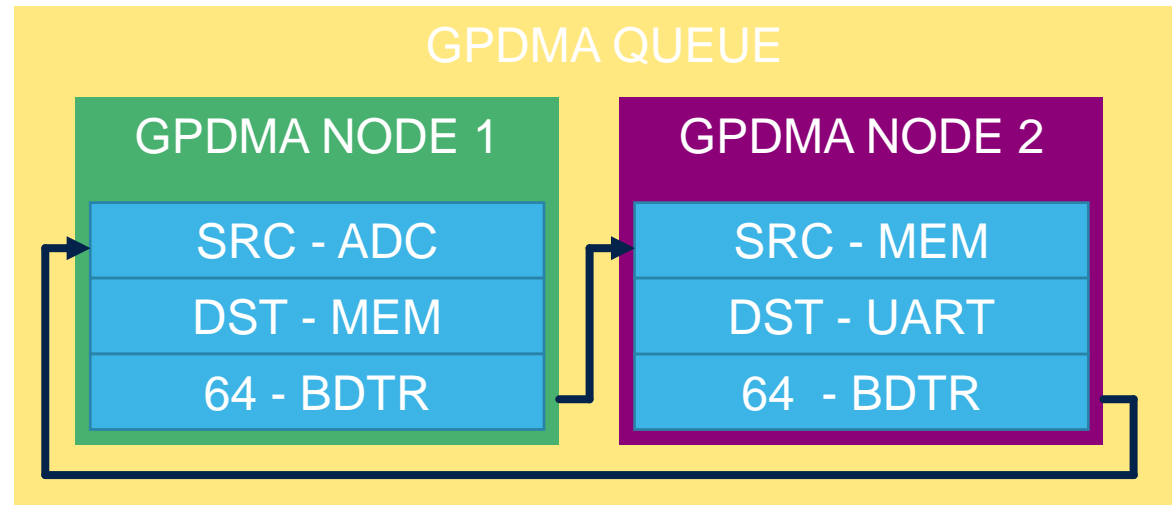
ADC + UART + TIM with LLI controlled GPDMA transfer



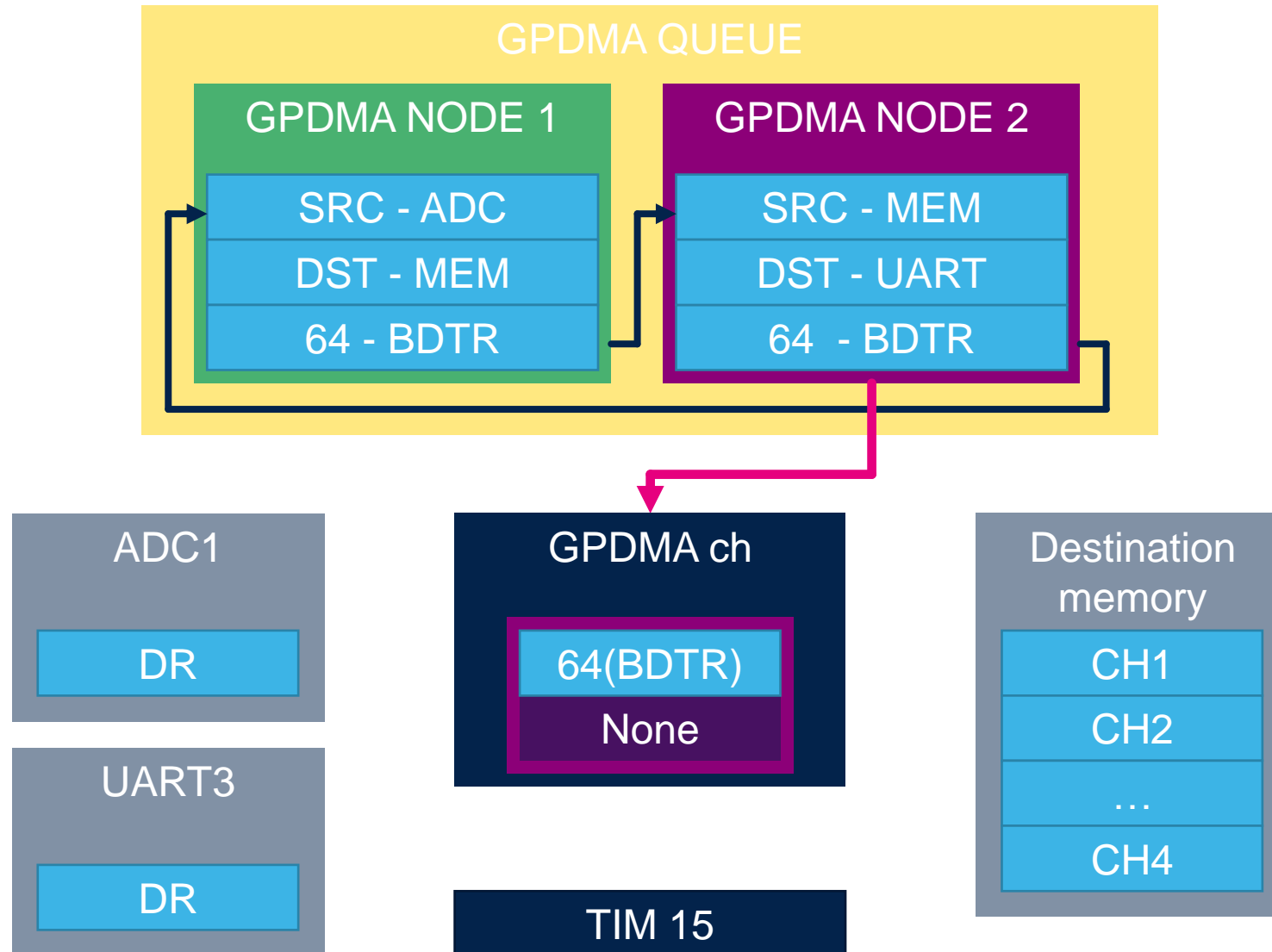
ADC + UART + TIM with LLI controlled GPDMA transfer



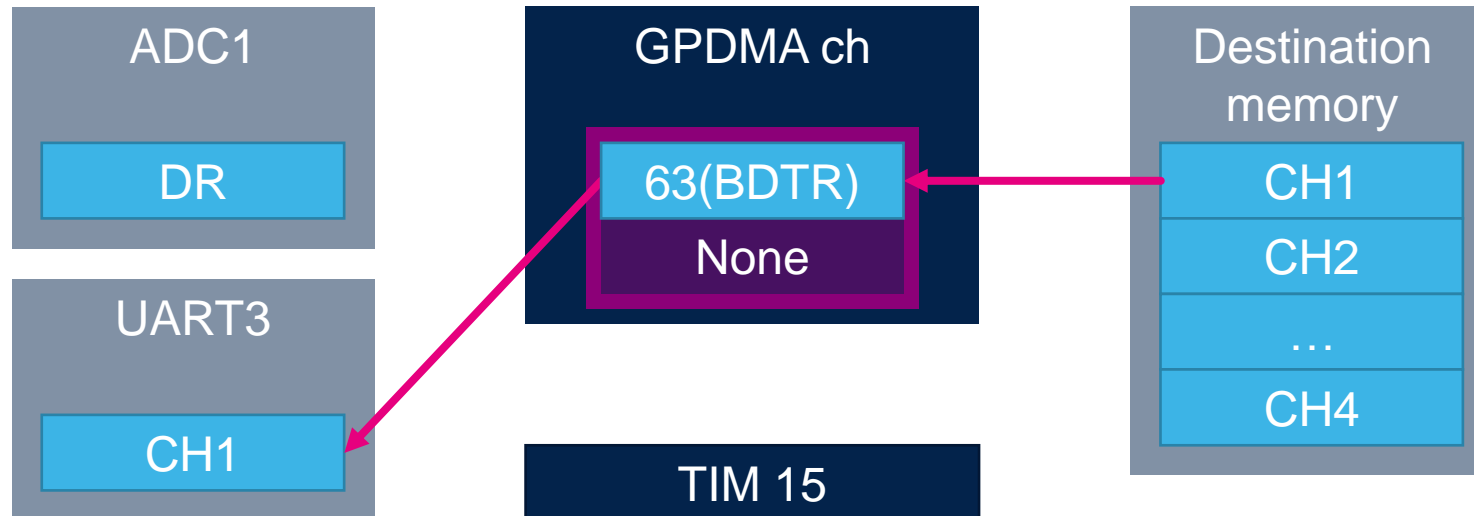
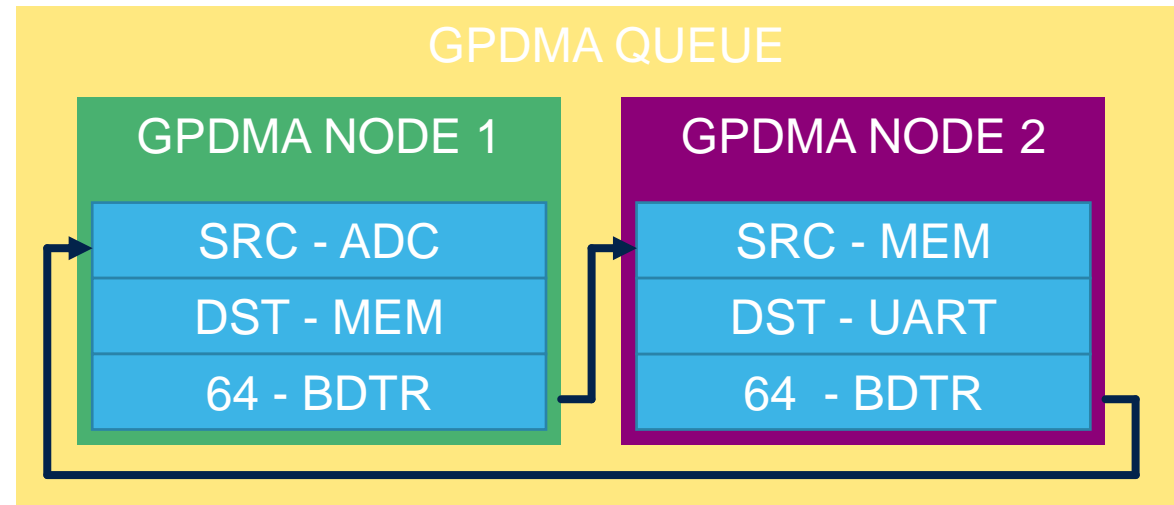
ADC + UART + TIM with LLI controlled GPDMA transfer



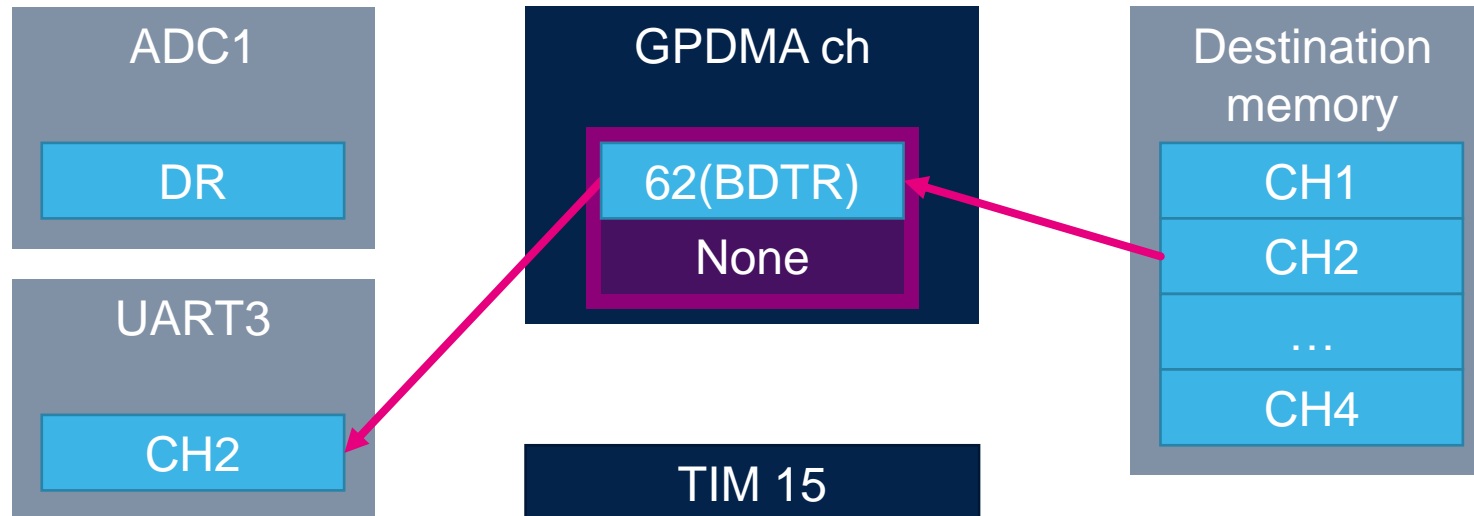
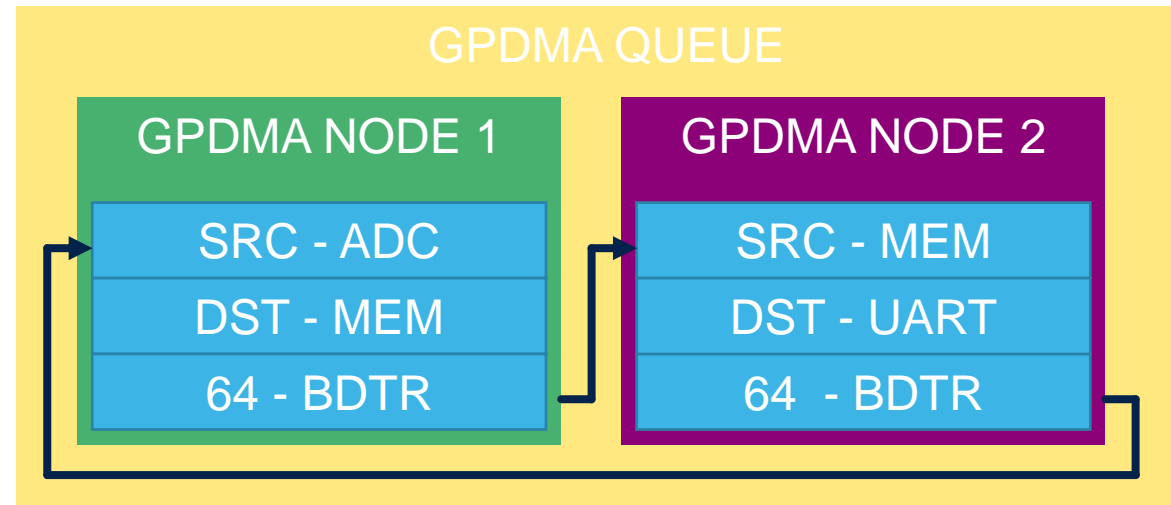
ADC + UART + TIM with LLI controlled GPDMA transfer



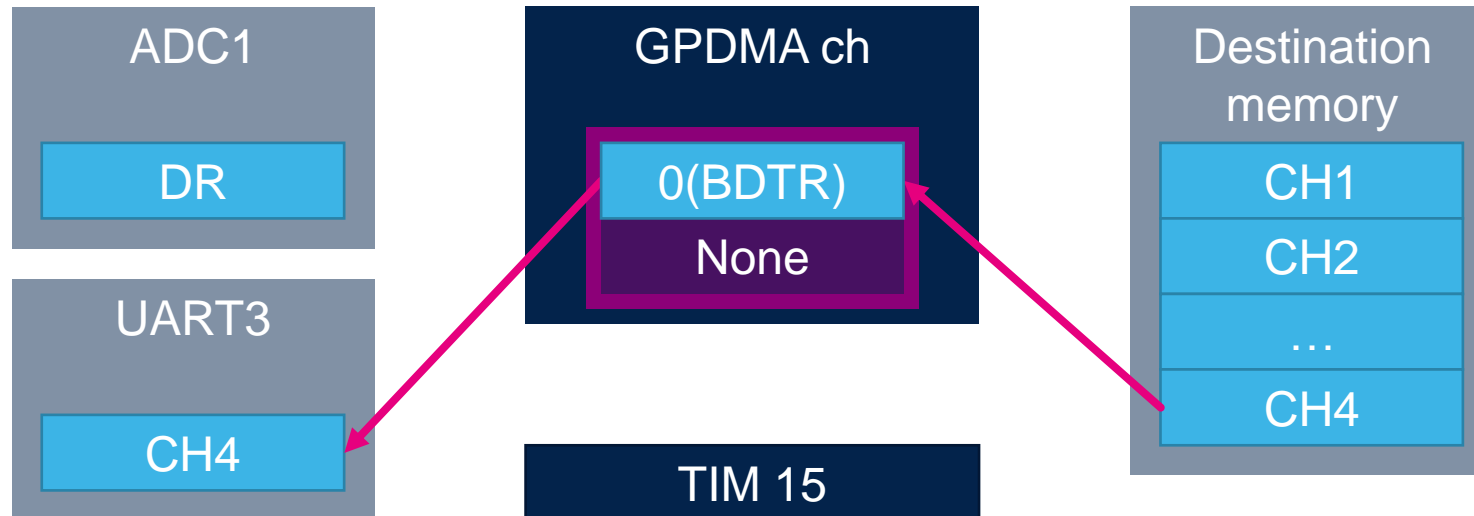
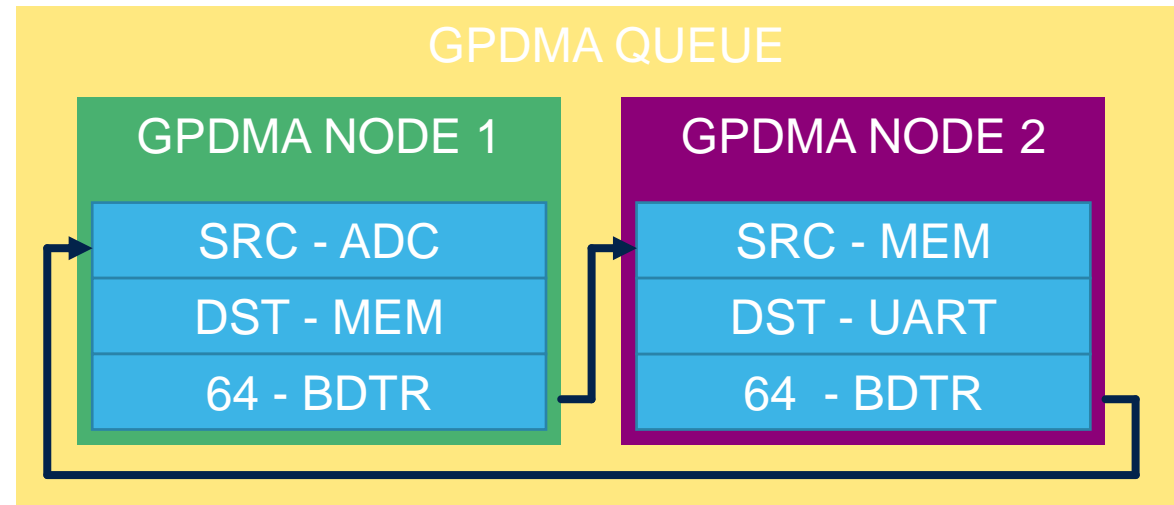
ADC + UART + TIM with LLI controlled GPDMA transfer



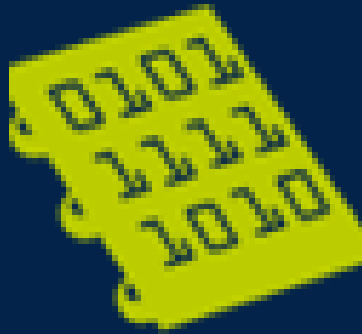
ADC + UART + TIM with LLI controlled GPDMA transfer



ADC + UART + TIM with LLI controlled GPDMA transfer



Hands-on



Please open STM32CubeIDE

Conclusion

Conclusion

- GPDMA implementation on H5
- GPDMA key features
- Autonomous data manipulation with no CPU load
- Standard request x Linked list based programming

What's next

Links to resources

- [STM32H563](#)
- [NUCLEO-H563ZI](#)
- [STM32CubeIDE](#)
- [STM32CubeMX](#)
- [STM32CubeProgrammer](#)
- [STM32 MCU Developer Zone](#)

ADC + UART + TIM + 2D – LLI GPDMA

The screenshot displays an IDE with the following components:

- Left Panel (Debug Console):** Shows the execution flow, including "H5_GPDMA Debug" and "Thread #1 [main] 1 [core: 1]".
- Center Panel (Code Editor):** Displays the C code for `main.c`. The code includes HAL DMA, UART, and TIM initialization, followed by a `while (1)` loop. The current execution point is at line 125, `while (1)`.
- Right Panel (Expressions):** Shows two memory inspection windows. The left window displays the `data` array (type `uint16_t [64]`) with values ranging from 359 to 749. The right window displays the `data2` array (type `uint16_t [64]`) with values ranging from 359 to 749. Arrows indicate the mapping of data between the two arrays.

Code Snippet (main.c):

```
111 HAL_DMAEx_List_LinkQ(&handle_GPDMA1_Channel6, &YourQueueName);
112
113 ATOMIC_SET_BIT(huart3.Instance->CR3, USART_CR3_DMAT);
114 __HAL_UART_ENABLE(&huart3);
115
116 HAL_DMAEx_List_Start(&handle_GPDMA1_Channel6);
117 ADC1->CFGR |= ADC_CFGR_DMAEN;
118 HAL_ADC_Start(&hadc1);
119
120 HAL_TIM_Base_Start(&htim15);
121 /* USER CODE END 2 */
122
123 /* Infinite loop */
124 /* USER CODE BEGIN WHILE */
125 while (1)
126 {
127     /* USER CODE END WHILE */
128
129     /* USER CODE BEGIN 3 */
130 }
131 /* USER CODE END 3 */
132 }
133
134 /**
135  * @brief System Clock Configuration
136  * @retval None
137  */
138 void SystemClock_Config(void)
139 {
140     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
141     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
142
143     /** Configure the main internal regulator output voltage
144     */
145     __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
146
147     while(!__HAL_PWR_GET_FLAG(PWR_FLAG_VOSRDY)) {}
148
149     /** Initializes the RCC Oscillators according to the specified
150     * in the RCC_OscInitTypeDef structure.
151     */
152     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_CSI;
153     RCC_OscInitStruct.CSIState = RCC_CSI_ON;
```

Memory Inspection Data:

| Expression | Type | Value |
|-----------------------|-----------------------|-------|
| <code>data[0]</code> | <code>uint16_t</code> | 749 |
| <code>data[1]</code> | <code>uint16_t</code> | 577 |
| <code>data[2]</code> | <code>uint16_t</code> | 578 |
| <code>data[3]</code> | <code>uint16_t</code> | 520 |
| <code>data[4]</code> | <code>uint16_t</code> | 359 |
| <code>data[5]</code> | <code>uint16_t</code> | 566 |
| <code>data[6]</code> | <code>uint16_t</code> | 583 |
| <code>data[7]</code> | <code>uint16_t</code> | 553 |
| <code>data[8]</code> | <code>uint16_t</code> | 391 |
| <code>data[9]</code> | <code>uint16_t</code> | 565 |
| <code>data[10]</code> | <code>uint16_t</code> | 592 |
| <code>data[11]</code> | <code>uint16_t</code> | 561 |
| <code>data[12]</code> | <code>uint16_t</code> | 524 |
| <code>data[13]</code> | <code>uint16_t</code> | 563 |
| <code>data[14]</code> | <code>uint16_t</code> | 607 |
| <code>data[15]</code> | <code>uint16_t</code> | 593 |
| <code>data[16]</code> | <code>uint16_t</code> | 423 |
| <code>data[17]</code> | <code>uint16_t</code> | 547 |
| <code>data[18]</code> | <code>uint16_t</code> | 597 |
| <code>data[19]</code> | <code>uint16_t</code> | 609 |
| <code>data[20]</code> | <code>uint16_t</code> | 471 |
| <code>data[21]</code> | <code>uint16_t</code> | 565 |
| <code>data[22]</code> | <code>uint16_t</code> | 607 |
| <code>data[23]</code> | <code>uint16_t</code> | 624 |
| <code>data[24]</code> | <code>uint16_t</code> | 599 |
| <code>data[25]</code> | <code>uint16_t</code> | 564 |
| <code>data[26]</code> | <code>uint16_t</code> | 608 |
| <code>data[27]</code> | <code>uint16_t</code> | 631 |
| <code>data[28]</code> | <code>uint16_t</code> | 547 |
| <code>data[29]</code> | <code>uint16_t</code> | 578 |
| <code>data[30]</code> | <code>uint16_t</code> | 607 |
| <code>data[31]</code> | <code>uint16_t</code> | 647 |
| <code>data[32]</code> | <code>uint16_t</code> | 503 |
| <code>data[33]</code> | <code>uint16_t</code> | 586 |

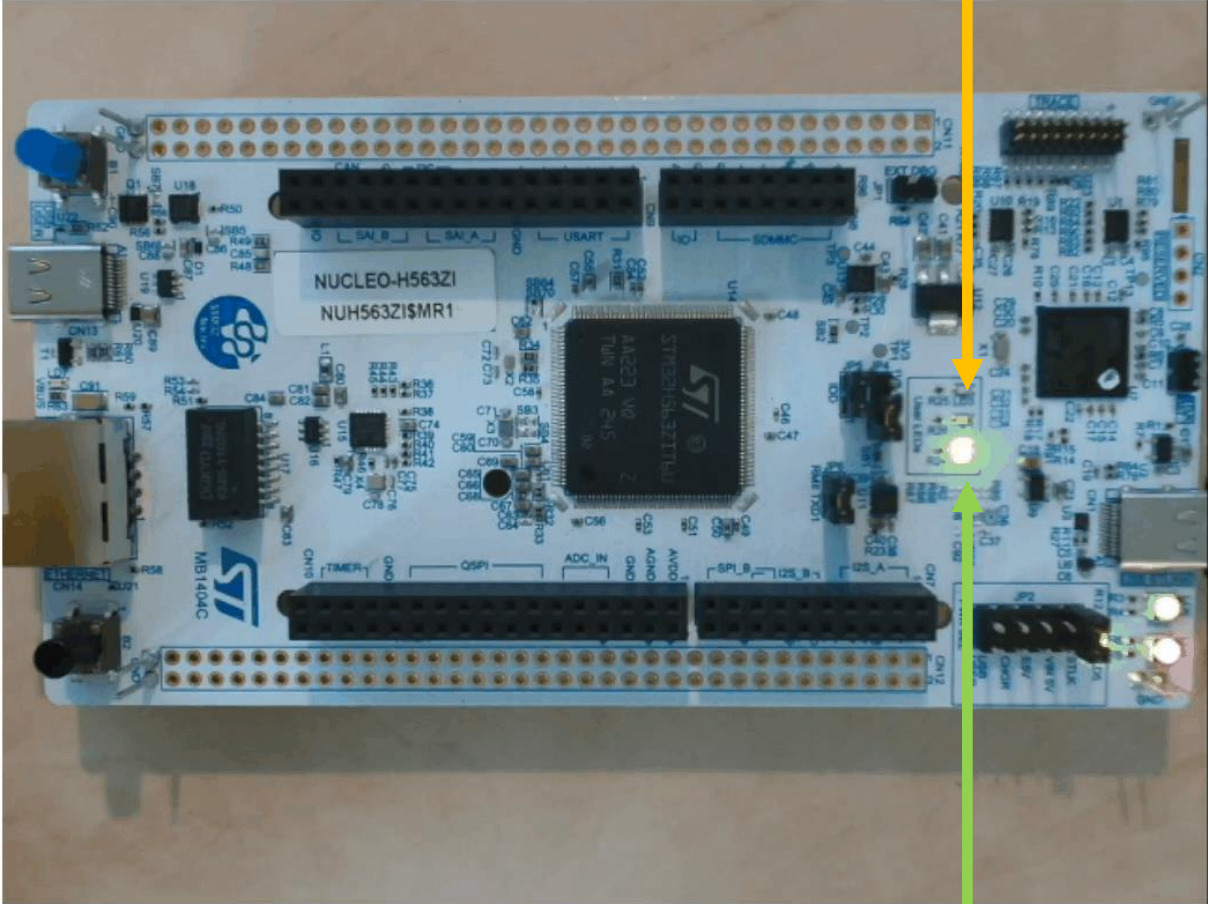
| Expression | Type | Value |
|------------------------|-----------------------|-------|
| <code>data2[0]</code> | <code>uint16_t</code> | 749 |
| <code>data2[1]</code> | <code>uint16_t</code> | 359 |
| <code>data2[2]</code> | <code>uint16_t</code> | 391 |
| <code>data2[3]</code> | <code>uint16_t</code> | 524 |
| <code>data2[4]</code> | <code>uint16_t</code> | 423 |
| <code>data2[5]</code> | <code>uint16_t</code> | 471 |
| <code>data2[6]</code> | <code>uint16_t</code> | 599 |
| <code>data2[7]</code> | <code>uint16_t</code> | 547 |
| <code>data2[8]</code> | <code>uint16_t</code> | 503 |
| <code>data2[9]</code> | <code>uint16_t</code> | 671 |
| <code>data2[10]</code> | <code>uint16_t</code> | 567 |
| <code>data2[11]</code> | <code>uint16_t</code> | 624 |
| <code>data2[12]</code> | <code>uint16_t</code> | 717 |
| <code>data2[13]</code> | <code>uint16_t</code> | 611 |
| <code>data2[14]</code> | <code>uint16_t</code> | 613 |
| <code>data2[15]</code> | <code>uint16_t</code> | 747 |
| <code>data2[16]</code> | <code>uint16_t</code> | 577 |
| <code>data2[17]</code> | <code>uint16_t</code> | 566 |
| <code>data2[18]</code> | <code>uint16_t</code> | 565 |
| <code>data2[19]</code> | <code>uint16_t</code> | 563 |
| <code>data2[20]</code> | <code>uint16_t</code> | 547 |
| <code>data2[21]</code> | <code>uint16_t</code> | 565 |
| <code>data2[22]</code> | <code>uint16_t</code> | 564 |
| <code>data2[23]</code> | <code>uint16_t</code> | 578 |
| <code>data2[24]</code> | <code>uint16_t</code> | 586 |
| <code>data2[25]</code> | <code>uint16_t</code> | 603 |
| <code>data2[26]</code> | <code>uint16_t</code> | 611 |
| <code>data2[27]</code> | <code>uint16_t</code> | 629 |
| <code>data2[28]</code> | <code>uint16_t</code> | 643 |
| <code>data2[29]</code> | <code>uint16_t</code> | 638 |
| <code>data2[30]</code> | <code>uint16_t</code> | 660 |
| <code>data2[31]</code> | <code>uint16_t</code> | 668 |
| <code>data2[32]</code> | <code>uint16_t</code> | 578 |

ADC + UART + TIM + 2D – LLI GPDMA

COM80 115200 bps, 8N1, no handshake

Settings Clear About Close

toggled by TC event



heartbeat in main loop