1 Section: Introducing the U
2 Section: Creating a pollec
3 Section: Queue-based dri
4 Section: A buffer-based d
5 Section: Configuring DM
6 Section: A buffer-based d
7 Section: Stream buffers (

Function/macro provider	Function/macro	Containing file
Arm	NVIC_EnableIRQ	Drivers\CMSIS\Include\core_cm7.h
Arm	NVIC_SetPriority	Drivers\CMSIS\Include\core_cm7.h
Arm	NVIC_SetPriorityGrouping	Drivers\CMSIS\Include\core_cm7.h
book	HWInit	BSP\Nucleo_F767ZI_Init.c
book	initUart2Pins	BSP\UartQuickDirtyInit.c
book	initUart4Pins	BSP\UartQuickDirtyInit.c
book	STM_UartInit	BSP\UartQuickDirtyInit.c
book	startReceiveInt	Chapter_10/Src/mainUartInterruptQueue.c
book	startUart4Traffic	Chapter_10/Src/mainUartInterruptQueue.c
book	uartPrintOutTask	Chapter_10/Src/mainUartInterruptQueue.c
book	USART2_IRQHandler	Chapter_10/Src/mainUartInterruptQueue.c
book	SetupUart4ExternalSim	Chapter_10\Src\Uart4Setup.c
book	uart4TxDmaSetup	Chapter_10\Src\Uart4Setup.c
book	uart4TxDmaStartRepeat	Chapter_10\Src\Uart4Setup.c
FreeRTOS	xQueueCreate	Middleware\Third_Party\FreeRTOS\Source\include\queue.h
FreeRTOS	xQueueSendFromISR	Middleware\Third_Party\FreeRTOS\Source\include\queue.h
FreeRTOS	xTimerStart	Middleware\Third_Party\FreeRTOS\Source\include\timers.h
FreeRTOS	portYIELD_FROM_ISR	Middleware\Third_Party\FreeRTOS\Source\portable\GCC\ARM_CM7\r0p1\portmacro.h
FreeRTOS	xQueueReceive	Middleware\Third_Party\FreeRTOS\Source\queue.c
FreeRTOS	vTaskStartScheduler	Middleware\Third_Party\FreeRTOS\Source\tasks.c
FreeRTOS	xTimerCreate	Middleware\Third_Party\FreeRTOS\Source\timers.c
SEGGER SystemView	SEGGER_SYSVIEW_PrintfHost	Middleware\Third_Party\SEGGER\SEGGER_SYSVIEW.c
SEGGER SystemView	SEGGER_SYSVIEW_RecordEnterISR	Middleware\Third_Party\SEGGER\SEGGER_SYSVIEW.c
SEGGER SystemView	SEGGER_SYSVIEW_RecordExitISR	Middleware\Third_Party\SEGGER\SEGGER_SYSVIEW.c
SEGGER SystemView	SEGGER_SYSVIEW_Conf	Third_Party\SEGGER\SEGGER_SYSVIEW_Config_FreeRTOS.c
STMicroelectronics/book	assert_param	Chapter_10\Inc\stm32f7xx_hal_conf.h
STMicroelectronics	HAL_RCC_DMA1_CLK_ENABLE	Drivers\STM32F7xx_HAL_Driver\Inc\stm32f7xx_hal_rcc.h
STMicroelectronics	HAL_NVIC_EnableIRQ	Drivers\STM32F7xx_HAL_Driver\Src\stm32f7xx_hal_cortex.c
STMicroelectronics	HAL_NVIC_SetPriority	Drivers\STM32F7xx_HAL_Driver\Src\stm32f7xx_hal_cortex.c
STMicroelectronics	HAL_GPIO_Init	Drivers\STM32F7xx_HAL_Driver\Src\stm32f7xx_hal_gpio.c
STMicroelectronics	HAL_UART_Init	Drivers\STM32F7xx_HAL_Driver\Src\stm32f7xx_hal_uart.c

- **Additional info**, page 254
  - In running `mainUartInterruptQueue.c`, the SystemView app reports overflow.
  - I attempted to prevent overflow by doing the following. However, overflow still occurred.
    - The baud-rate was reduced from 256,400 to 150 (in `mainUartInterruptQueue.c`).
      - SystemView was used to measure throughput. For a specified baud-rate of 150, the actual baud-rate was 700 (bytes-received/second). This measurement's accuracy was not affected by the SystemView overflow.
    - `SEGGER_SYSVIEW_RTT_BUFFER_SIZE` was increased from 10,240 to 32,000, in `SEGGER_SYSVIEW_Conf.h`. (I assumed that extra memory is available, but I don't know if it is.)
    - In `uartPrintOutTask()`, code was added so `SEGGER_SYSVIEW_PrintfHost()` is issued once for every 2,000 receives.
  - For fixing SystemView overflow, general techniques are presented in the study-guide's SystemView page.

- **Additional info**, page 254
  - I ran some experiments to see if the actual baud-rate is the same as the specified baud-rate.
  - Several baud-rates were tried using `mainUartInterruptQueue.c`. The program was modified to call `SEGGER_SYSVIEW_PrintfHost()` for every 2,000 characters received.
  - The actual throughput was close to the specified baud-rate, for baud-rates 9,600 and 256,400. For a specified baud-rate of 150, the actual throughput was much more. It was assumed that 10 bits are sent for each character (includes start and stop bits).

Baud rate	Characters/Sec	Bits/Sec
150	700	7,000
9,600	967	9,670
256,400	25,394	253,940