LFX Coding Challenge: DMA-based UART Communication Implementation.

This file explains the implementation of the reusable modular drivers developed to configure and manage the UART communication using the DMA for efficient data transmission and reception. The codes were implemented using bare-metal embedded C programming which refers to writing code directly for the hardware of a microcontroller or embedded system using C programming language. The Nucleo-F401RE board of the STM32F4 microcontroller family is used to design and implement the system. The USART2 port (PA2=TX, and PA3=RX) is configured to perform the DMA-based UART Communication.

The codes consist in:

- **Main:** The DMA-based UART communication is implemented.
- **Digital Input Output driver (DIO):** A group of codes used to configure the GPIOs used for the UART communication.
- **Universal Synchronous/Asynchronous receiver transmitter driver (USART):** A group of codes used to configure the UART communication.
- **Direct Memory Access Driver (DMA):** A group of codes used to configure the DMA system.

The drivers' codes are structured as follows:

- driverName_cfg.h: contains interface definitions used to configure the driver register.
- driverName_cfg.c: configure the configuration array used to set the register map.
- driverName.h: contains the definitions of the interface for the peripheral to be configured.
- driverName.c: configure the register map for the peripheral.

The code explanation will be performed according to the main sequence of the application. The main code starts including the libraries created to implement the drivers:

```
/*****************************************************************************
* Includes
*****************************************************************************/
#include<stdio.h>
#include<stdint.h>
#include "usart.h"
#include "dio.h"
#include "dma.h"
```

Figure 1. Libraries.

The following part shows the clock configuration for GPIOA, USART2, and DMA1 to enable the peripheral clocks.

```
int main(void)
{   /*Enable clock access to GPIOA, USART2, and DMA1*/
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
    RCC->APB1ENR |= RCC_APB1ENR_USART2EN;
    RCC->AHB1ENR |= RCC_AHB1ENR_DMA1EN;
```

Figure 2. Enable Peripheral Clocks.

The next line is a struct array that retrieves a configuration table for the DIO (GPIO) driver. The USART and DMA drivers also use the same methodology later.

```
/*Get the address of the configuration table for DIO*/
const DioConfig_t * const DioConfig = DIO_configGet();
```

Figure 3. Configuration array.

Configuration files.

Let's analyze the contents of dio_cfg.c to understand its implementation and header file dio_cfg.h to see how the configuration table is filled. The header file defines the interface for configuring GPIO pins on the microcontroller. It provides constants, enumerations, and structures to manage the GPIO settings, ensuring flexible configuration. The header files of the USART (usart_cfg.h), and DMA (dma_cfg.h) perform the same functionality.

The dio_cfg.h defines:

1. Port and Pin Definitions:
    - Defines five GPIO ports (PA, PB, PC, PD, PH).
    - Enumerates all pins for each port, allowing easy reference.

2. Pin Configuration Options:
    - Modes: Input, Output, Alternate Function, Analog.
    - Types: Push-pull or Open-drain output.
    - Speed Settings: Low, Medium, High, Very High.
    - Resistor Configuration: No resistor, Pull-up, Pull-down.
    - Alternate Functions: Multiplexing options for peripherals.
3. Configuration Structure (DioConfig_t):
    - Defines all necessary parameters for configuring a GPIO pin.
    - Includes port, pin, mode, type, speed, resistor settings, and function selection.
4. Retrieving Configuration (DIO_configGet()):
    - Provides a function to retrieve predefined pin configurations, used in system initialization.

The usart_cfg.h defines:

1. USART Peripheral Configuration:
    - Supports three USART ports (USART1, USART2, USART6).
    - Defines active USART ports (USART_USED_PORTS).
2. USART Communication Settings:
    - Data Word Length: 8-bit or 9-bit data format.
    - Stop Bits: 1, 0.5, 2, or 1.5 stop bits.
    - Parity: Enabled (even) or Disabled (no parity).
    - Baud Rate Options: Common standard baud rates (9600, 19200, 38400, etc.).
3. USART Transmission & Reception Control:
    - RX and TX Modes: Enables/disables receive (RX) and transmit (TX) functionality.
    - DMA Support: Configures DMA (Direct Memory Access) for TX and RX for efficient data transfer.
4. USART Configuration Structure (UsartConfig_t).
    - Stores USART settings, including port, word length, stop bits, parity, TX/RX enablement, DMA support, and baud rate.
5. Retrieving Configuration (USART_configGet()).
    - Provides a function to retrieve predefined USART configurations, used in system initialization.

The dma_cfg.h defines:

1. DMA Stream and Channel Configuration:
    - Supports 16 DMA streams (DMA1_STREAM_0 to DMA2_STREAM_7).
    - Defines 8 DMA channels (DMA_CHANNEL_0 to DMA_CHANNEL_7).
    - Allows selection of used DMA ports (DMA_USED_PORTS).
2. DMA Transfer Direction:
    - Peripheral to Memory (e.g., ADC data to RAM).
    - Memory to Peripheral (e.g., RAM data to UART).
    - Memory to Memory (e.g., internal RAM data transfer).
3. Data Transfer Size & Increment Modes:
    - Supports 8-bit, 16-bit, and 32-bit transfers for both memory and peripheral.
    - Configurable increment mode for memory and peripheral (enabled/disabled).
4. FIFO and Direct Mode Settings:
    - Enables/disables FIFO direct mode for optimized data handling.
    - Configurable FIFO threshold levels (1/4, 1/2, 3/4, Full).
5. DMA Configuration Structure (DmaConfig_t).
    - Defines parameters for stream, channel, direction, data size, increment mode, FIFO mode, and threshold.
6. Retrieving Configuration (DMA_configGet()).
    - Provides a function to retrieve predefined DMA configurations, used in initialization.

The dio_cfg.c, dma_cfg.c, and usart_cfg.c files implement the configuration for each peripheral. It provides a structured approach to configuring the peripheral using a predefined table and a function to retrieve the configuration.
The dio_cfg.c implements:

1. Configuration Table (DioConfig, UsartConfig, and DmaConfig).
    - Defines a constant array containing the settings for the peripheral (GPIO, USART, and DMA).
    - Each row represents a single pin, port, and stream respectively, and each column represents a configuration parameter.
        - GPIO parameters include:
            - Port & Pin (e.g., DIO_PA, DIO_PA2).

        - Mode (e.g., DIO_FUNCTION).
        - Output Type (e.g., DIO_PUSH_PULL).
        - Speed (e.g., DIO_LOW_SPEED).
        - Pull-up/Pull-down Resistor (e.g., DIO_PULLUP).
        - Alternate Function Selection (e.g., DIO_AF7).
    - USART parameters include:
        - Port (e.g., USART_PORT_2).
        - Word Length & Stop Bits (e.g., USART_WORD_LENGTH_8, USART_STOP_BITS_1).
        - Parity Mode (e.g., USART_PARITY_DISABLED).
        - RX & TX Modes (e.g., USART_RX_ENABLED, USART_TX_ENABLED).
        - DMA Configuration (e.g., USART_RX_DMA_ENABLED, USART_TX_DMA_ENABLED).
        - Baud Rate (e.g., USART_BAUD_RATE_9600).
    - DMA parameters include:
        - Stream & Channel (e.g., DMA1_STREAM_6, DMA_CHANNEL_4).
        - Transfer Direction (e.g., DMA_MEMORY_TO_PERIPHERAL).
        - Memory & Peripheral Data Size (e.g., DMA_MEMORY_SIZE_8, DMA_PERIPHERAL_SIZE_8).
        - Increment Modes (e.g., DMA_MEMORY_INCREMENT_ENABLED).
        - FIFO Mode & Threshold (e.g., DMA_FIFO_DIRECT_MODE_ENABLED, DMA_FIFO_THRESHOLD_FULL).
2. Retrieving Configuration (DIO_configGet(), USART_configGet(), and DMA_configGet ()):
    - Returns a pointer to the first element of the DioConfig array, UsartConfig array or DmaConfig array.
    - Ensures read-only access to prevent modifications to the configuration table.
    - Used in DIO_Init(), USART_Init(), and DMA_Init() to initialize the peripherals based on predefined settings.

Implementation files.

Let's analyze the contents of dio.c to understand its implementation and header file dio.h to see how the driver (peripheral) is configured. The header file defines the function's prototype to configure and handle GPIO pins on the microcontroller. The header files of the USART (usart.h), and DMA (dma.h) perform the same functionality.

The headers files define:

1. Included Headers:
    - dio_cfg.h: Provides DIO configuration settings.
    - stm32f4xx.h: Includes hardware-specific definitions for the STM32F4 family (Registers map).
    - Standard headers like <stdint.h> and <stdio.h>.
2. Preprocessor Constants:
    - Defines error codes, such as DIO_ERROR_CODE_NONE (indicating no errors).
3. Function Prototypes:
    a. dio.c
        - DIO_init(const DioConfig_t * const Config): Initializes the DIO module using a configuration table.
        - DIO_pinRead(DioPort_t Port, DioPin_t Pin): Reads the state of a specified pin.
        - DIO_pinWrite(DioPort_t Port, DioPin_t Pin, DioPinState_t State): Writes a HIGH or LOW state to a pin.
        - DIO_pinToggle(DioPort_t Port, DioPin_t Pin): Toggles the current state of a pin.
        - DIO_registerWrite(uint32_t address, uint32_t value): Writes a value to a specific hardware register.
        - DIO_registerRead(uint32_t address): Reads a hardware register value.
    b. usart.c
        - USART_init(const UsartConfig_t * const Config, const uint32_t peripheralClock): Initializes the USART peripheral with the specified configuration and clock speed.
        - USART_transmit(const UsartPort_t Port, const char * const data): Sends a string of characters over the specified USART port.
        - USART_receive(const UsartPort_t Port, char * const data): Receives a string of characters from the specified USART port.
        - USART_registerWrite(const uint32_t address, const uint32_t value): Writes a value to a USART hardware register.
        - USART_registerRead(const uint32_t address): Reads a USART hardware register value.
    c. Dma.c
        - DMA_init(const DmaConfig_t * const Config);
            i. Initializes the DMA peripheral based on the specified configuration table.

- DMA_transferConfig(const DmaStream_t Stream, volatile uint32_t * const peripheral, const uint32_t * memory, const uint32_t length);
  ii. Configures a DMA transfer between a memory location and a peripheral.
  iii. Parameters:
    1. Stream: The DMA stream is being configured.
    2. peripheral: Pointer to the peripheral register.
    3. memory: Pointer to the memory source/destination.
    4. length: Number of data units to be transferred.

4. C++ Compatibility: Uses extern "C" to allow usage in C++ projects.

The dio.c, usart.c, and dma.c files implement the functions of each peripheral. It provides arrays with the MCU register, an enumeration for error codes algorithm, and functions to configure and handle the peripherals.

```
/*Initialize the DIO pins according to the configuration table*/
DIO_init(DioConfig);
```

Figure 4. Initialization function call.

DMA Transfer configuration.

The following lines are used to configure the Direct Memory Access (DMA) for USART2 data transmission (Tx) and reception (Rx) using DMA1.

```
/*Configure the DMA peripheral (USART_TX) for a transfer to memory*/
DMA_transferConfig(DMA1_STREAM_6, (uint32_t*)&USART2->DR, (uint32_t*)&txBuffer[0], sizeof(txBuffer));

/*Configure the DMA peripheral (USART_RX) for receiving data from memory*/
DMA_transferConfig(DMA1_STREAM_5, (uint32_t*)&USART2->DR, (uint32_t*)&rxBuffer, sizeof(rxBuffer));
```

Figure 5. Transfer Configuration Functions.

The first function call configures the DMA1(stream 6) to transfer data from memory (txBuffer) to USART's data register (USART2->DR), enabling direct data transmission without CPU intervention.
The second function call configures the DMA1(stream5) to transfer data from USART2's data register (USART2->DR) into a memory buffer (rxBuffer).

Application Purpose.

The setup allows efficient, interrupt-driven USART communication without CPU overhead by offloading data movement to the DMA controller.