Console_Driver

V 1.0.0

By Kunal Salvi

Chapter 1

Console Module for STM32F407VGT6

1.1 Introduction

The Console module provides an abstraction layer for UART communication on the STM32F407VGT6 microcontroller. It is designed for efficient data handling using DMA and provides easy-to-use functions for formatted printing and reading through a serial interface.

1.2 Features

- UART initialization with custom baud rate
- Formatted printing using `printConsole`
- Formatted input using `readConsole`
- DMA-based UART reception for high efficiency
- Designed to support common debugging and communication tasks

1.3 Dependencies

The Console module depends on the following components:

- GPIO for UART pin configuration
- USART for UART initialization and communication
- DMA for efficient UART data handling

Include the following headers in your project to use the Console module:

- `Console.h`
- `GPIO.h` (for GPIO configuration)
- `USART.h` (for UART initialization and communication)
- `DMA.h` (for DMA configuration)

1.4 Usage

To use the Console module, include the `Console.h` header in your application code. Initialize the console with a desired baud rate using `Console_Init`. Use `printConsole` for sending formatted messages and `readConsole` for receiving formatted input.

Example:
```
#include "Console.h"

int main(void) {
    Console_Init(9600); // Initialize UART with 9600 baud rate

    printConsole("Hello, STM32!\n"); // Send a message

    char buffer[20];
    readConsole("%s", buffer); // Read user input into buffer

    printConsole("You entered: %s\n", buffer); // Echo the input

    while (1) {
        // Application code
    }
}
```

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

File Documentation

3.1 Console.c

```c
#include "Console.h"

// Flags to control and monitor UART reception
volatile int rx_get_flag = 0; // Indicates if the reception is active
volatile int rx_flag = 0;     // Indicates if data reception is complete

#define RX_Buffer_Length 100 // Length of the reception buffer

// Variables to track the length of received data and the reception buffer
volatile int RX_Length = 0;
volatile char TRX_Buffer[RX_Buffer_Length]; // Buffer for received and transmitted data

// USART configuration structure
USART_Config serial;



void UART4_IRQHandler(void) {
    if (rx_get_flag == 1) { // Check if reception is active
        (void)UART4->SR; // Read the status register to clear flags
        (void)UART4->DR; // Read the data register to clear flags

        __disable_irq(); // Disable interrupts to safely update DMA configurations

        // Disable DMA stream
        serial.USART_DMA_Instance_RX.Request.Stream->CR &= ~DMA_SxCR_EN;

        // Calculate the length of received data
        RX_Length = RX_Buffer_Length - serial.USART_DMA_Instance_RX.Request.Stream->NDTR;

        // Prevent buffer overflow
        if (RX_Length > RX_Buffer_Length) {
            RX_Length = RX_Buffer_Length;
        }

        // Reset DMA stream for the next reception
        serial.USART_DMA_Instance_RX.Request.Stream->NDTR = RX_Buffer_Length;
        serial.USART_DMA_Instance_RX.Request.Stream->CR |= DMA_SxCR_EN;

        __enable_irq(); // Re-enable interrupts

        rx_flag = 1; // Set the flag indicating data reception is complete
    }
}


void Console_Init(int32_t baudrate) {
    // Reset USART configuration to default values
    USART_Config_Reset(&serial);

    // Configure USART parameters
    serial.Port = UART4; // Use UART4 for console communication
    serial.baudrate = baudrate; // Set the baud rate
    serial.mode = USART_Configuration.Mode.Asynchronous; // Asynchronous mode
    serial.stop_bits = USART_Configuration.Stop_Bits.Bit_1; // 1 stop bit
    serial.TX_Pin = UART4_TX_Pin.PC10; // TX pin is PC10
    serial.RX_Pin = UART4_RX_Pin.PC11; // RX pin is PC11
    serial.interrupt = USART_Configuration.Interrupt_Type.IDLE_Enable; // Enable IDLE interrupt
    serial.dma_enable = USART_Configuration.DMA_Enable.TX_Enable | USART_Configuration.DMA_Enable.RX_Enable; // Enable DMA for TX and RX

    // Initialize USART
    if (USART_Init(&serial) != true) {
        // Handle USART initialization failure (e.g., log error or halt execution)
    }
}


void printConsole(char *msg, ...) {
    va_list args;
    va_start(args, msg);

    // Format the message and store it in the transmission buffer
    vsprintf((char *)TRX_Buffer, msg, args);

    // Get the length of the formatted string
    uint16_t len = strlen((char *)TRX_Buffer);

    // Transmit the buffer using DMA
    USART_TX_Buffer(&serial, (uint8_t *)&TRX_Buffer[0], len);

    va_end(args);
}




int readConsole(const char *msg, ...) {
    va_list args;
    int result;

    rx_get_flag = 1; // Enable reception

    // Start DMA reception
    USART_RX_Buffer(&serial, (uint8_t *)TRX_Buffer, RX_Buffer_Length, 0);

    // Wait until data reception is complete
    while (rx_flag == 0) {
        // Wait loop
    }

    // Check for valid input length
    if (RX_Length < 2) {
        // Reset flags and return error
        rx_get_flag = 0;
        rx_flag = 0;
        return -1;
    }

    // Null-terminate the received string
    TRX_Buffer[RX_Length - 1] = '\0';

    // Parse the input using the format string
    va_start(args, msg);
    result = vsscanf((char *)TRX_Buffer, msg, args);
    va_end(args);

    // Reset reception flags
    rx_get_flag = 0;
    rx_flag = 0;

    return result;
}
```

3.2 Console.h File Reference

Console Interface for STM32F407VGT6.

```c
#include "main.h"
#include "GPIO/GPIO.h"
#include "USART/USART.h"
#include "DMA/DMA.h"
```

Include dependency graph for Console.h:

3.3 Console.h

Go to the documentation of this file.

```
#ifndef CONSOLE_H_
#define CONSOLE_H_

#include "main.h"
#include "GPIO/GPIO.h"
#include "USART/USART.h"
#include "DMA/DMA.h"

void Console_Init(int32_t baudrate);

void printConsole(char *msg, ...);

int readConsole(const char *msg, ...);

#endif /* CONSOLE_H_ */

```

Index