```c
/**
 * @file: uart_driver.h
 * @author: Grant Wilk
 * @modified: 2/4/2020
 * @description: a UART driver for the MCU's USART2 module
*/

/**
 * Initializes USART2 as a UART
 * @param baud - the baud rate
 * @param sysclk - the frequency of the system clock in Hz
 */
void uart_init(int baud, int sysclk);
```

```c
/**
 * @file: uart_driver.c
 * @author: Grant Wilk
 * @modified: 2/4/2020
 * @description: a UART driver for the MCU's USART2 module
 */

# include <stdio.h>
# include "uart_driver.h"
# include "circular_queue.h"
# include "stm32f446xx.h"

/**
 * Stores characters after they are read from the RDR
 */
static circular_queue input_buffer;


/**
 * Stores characters until they are transmitted by the TDR
 */
static circular_queue output_buffer;

/**
 * Initializes USART2 as a UART
 * @param baud - the baud rate
 * @param sysclk - the frequency of the system clock in Hz
 */
void uart_init(int baud, int sysclk) {

    // define the input and output buffer
    input_buffer = cq_init();
    output_buffer = cq_init();

    // enable GPIOA in RCC
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

    // enable USART2 in RCC
    RCC->APB1ENR |= RCC_APB1ENR_USART2EN;

    // set PA2 and PA3 as pullup
    GPIOA->PUPDR |= (0b01 << GPIO_PUPDR_PUPD2_Pos | 0b01 << GPIO_PUPDR_PUPD3_Pos
);

    // clear PA2 and PA3 mode
    GPIOA->MODER &= ~(GPIO_MODER_MODER2 | GPIO_MODER_MODER3);

    // set PA2 and PA3 mode to alternate function
    GPIOA->MODER |= (0b10 << GPIO_MODER_MODER2_Pos | 0b10 <<
GPIO_MODER_MODER3_Pos);

    // clear alternate function select for PA2 and PA3
    GPIOA->AFR[0] &= ~(GPIO_AFRL_AFRL2 | GPIO_AFRL_AFRL3);

    // select USART1..3 (AF7) as the alternate function for PA3 and PA2
    GPIOA->AFR[0] |= (7 << GPIO_AFRL_AFSEL2_Pos | 7 << GPIO_AFRL_AFSEL3_Pos);

```

```
56      // set USART2's baud rate
57      USART2->BRR = sysclk / baud;
58
59      // enable USART2's UART, RX, and TX
60      USART2->CR1 |= (USART_CR1_UE | USART_CR1_TE | USART_CR1_RE);
61
62      // enable USART2's TXE interrupt and RXNE interrupt
63      USART2->CR1 |= (USART_CR1_TXEIE | USART_CR1_RXNEIE);
64
65      // enable USART2 interrupts in NVIC
66      NVIC->ISER[1] |= (1 << 6);
67
68      // set output buffer source
69      setvbuf(stdout, NULL, _IONBF, 0);
70
71  }
72
73  /**
74   * Reads a string from the UART's input buffer
75   * @param file - not implemented (ignored)
76   * @param ptr - where the read data should be put
77   * @param len - the number of characters to read
78   * @return the number of characters read
79   */
80  int _read(int file, char * ptr, int len) {
81
82      // wait until the input buffer receives some data
83      while (cq_isempty(&input_buffer));
84
85      int char_count = 0;
86
87      // pull from the circular queue until it is empty
88      while (!cq_isempty(&input_buffer)) {
89          char_count++;
90          *ptr = cq_pull(&input_buffer);
91          ptr++;
92      }
93
94      if (*ptr == '\r') *ptr = '\n';
95
96      return char_count;
97
98  }
99
100 /**
101  * Writes a string to the UART's output buffer
102  * @param file - not implemented (ignored)
103  * @param ptr - where the characters should be read from
104  * @param len - the number of characters to read
105  * @return the number of characters read
106  */
107 int _write(int file, char * ptr, int len) {
108
109     int char_count = 0;
110
111     // push characters to the output buffer until we write len characters or
    the buffer fills up
```

```
112     while (char_count < len && !cq_isfull(&output_buffer)) {
113         cq_push(&output_buffer, *ptr);
114         char_count++;
115         ptr++;
116     }
117 
118     // enable TXE interrupts so the data can be transmitted
119     USART2->CR1 |= USART_CR1_TXEIE;
120 
121     return char_count;
122 }
123 
124 /**
125  * USART2 interrupt request handler
126  */
127 void USART2_IRQHandler(void) {
128 
129     // if the RDR has received data and the input buffer is not full
130     if ((USART2->SR & USART_SR_RXNE) && !cq_isfull(&input_buffer)) {
131 
132         // read the RDR
133         char c = USART2->DR;
134 
135         // push the char in the RDR into the input buffer
136         cq_push(&input_buffer, c);
137 
138         // echo the character to the output buffer
139         if (!cq_isfull(&output_buffer)) {
140             cq_push(&output_buffer, c);
141         }
142 
143         // enable TXE interrupts so the echo can be pushed
144         USART2->CR1 |= USART_CR1_TXEIE;
145 
146     }
147 
148     // if the TDR has completed transmission
149     else if (USART2->SR & USART_SR_TXE) {
150 
151         // if the output buffer is not empty
152         if (!cq_isempty(&output_buffer)) {
153 
154             // pull a char out of the output buffer and move it into the TDR
155             USART2->DR = cq_pull(&output_buffer);
156 
157         } else {
158 
159             // disable TXE interrupts
160             USART2->CR1 &= ~(USART_CR1_TXEIE);
161 
162         }
163     }
164 }
165 
```