```c
#include <stdint.h>
#include "eecs388_lib.h"

void gpio_mode(int gpio, int mode)
{
  uint32_t val;

  if (mode == OUTPUT) {
    val = *(volatile uint32_t *) (GPIO_CTRL_ADDR + GPIO_OUTPUT_EN);
    val |= (1<<gpio);
    *(volatile uint32_t *) (GPIO_CTRL_ADDR + GPIO_OUTPUT_EN) = val;

    if (gpio == RED_LED || gpio == GREEN_LED || gpio == BLUE_LED) {
      // active high
      val = *(volatile uint32_t *) (GPIO_CTRL_ADDR + GPIO_OUTPUT_XOR);
      val |= (1<<gpio);
      *(volatile uint32_t *) (GPIO_CTRL_ADDR + GPIO_OUTPUT_XOR) = val;
    }
  } else if (mode == INPUT) {
    val = *(volatile uint32_t *) (GPIO_CTRL_ADDR + GPIO_INPUT_EN);
    val |= (1<<gpio);
    *(volatile uint32_t *) (GPIO_CTRL_ADDR + GPIO_INPUT_EN) = val;
  }
  return;
}

void gpio_write(int gpio, int state)
{
  uint32_t val = *(volatile uint32_t *) (GPIO_CTRL_ADDR + GPIO_OUTPUT_VAL);
  if (state == ON)
    val |= (1<<gpio);
  else
    val &= (~(1<<gpio));
  *(volatile uint32_t *) (GPIO_CTRL_ADDR + GPIO_OUTPUT_VAL) = val;
  return;
}

inline uint64_t get_cycles(void)
{
  return *(volatile uint64_t *)(CLINT_CTRL_ADDR + CLINT_MTIME);
}

void delay(int msec)
{
  uint64_t tend;
  tend = get_cycles() + msec * 32768 / 1000;
  while (get_cycles() < tend) {};
}

/**
 * write a chacter string to the UART 0
 *
 * Input:
 *  @str    string point
 * Return: None
 */
void ser_printline(char *str)
{
  int i;
  for (i = 0;; i++) {
    if (str[i] == '\0') {
      ser_write('\n');
      break;
    }
    ser_write(str[i]);
  }
}

void ser_setup()
{
  /* initialize UART0 TX/RX */
  *(volatile uint32_t *)(UART0_CTRL_ADDR + UART_TXCTRL) |= 0x1;
  *(volatile uint32_t *)(UART0_CTRL_ADDR + UART_RXCTRL) |= 0x1;
}

/**
 * write a chacter to the UART 0 FIFO
 *
 * Input:
 *  @c    character to send via the UART
 * Return: None
 */
void ser_write(char c)
{
  uint32_t regval;
  /* busy-wait if tx FIFO is full  */
  do {
    regval = *(volatile uint32_t *)(UART0_CTRL_ADDR + UART_TXDATA);
  } while (regval & 0x80000000);

  /* write the character */
  *(volatile uint32_t *)(UART0_CTRL_ADDR + UART_TXDATA) = c;
}

/**
 * read a chacter from the uart 0
 *
 * Input: None
 * Return: read byte
 */
char ser_read()
{
  /*
    The provided implementation doesn't actually read from the UART 0.
    The character 'r' is hardcoded to return.

    What you need to do to implement this function are:
    1) wait until UART0 RX FIFO is not empty
    2) read the data from the FIFO and return the read (one) byte.
  */

  uint32_t regval;

  do {
    regval = *(volatile uint32_t *)(UART0_CTRL_ADDR + UART_RXDATA);
  } while (regval & 0x80000000);

  return (uint8_t)(regval);
```

```
}
```