```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void ToggleLEDPin(uint32_t PinNo);
void DisableLEDPin(uint32_t PinNo);
void EnableLEDPin(uint32_t PinNo);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */
```

```
static char New = 0;
static char Color = 'o';
static char Value = '0';

void WipeColor()
{
	New = 0;
	Color = 'o';
	Value = '0';
}

void SetColor(char c, char val)
{
	Color = c;
	Value = val;
	New = 1;
}

char GetColor()
{
	return Color;
}

char GetStatus()
{
	return Value;
}

void InitLEDGPIOPin(uint32_t PinIndex)
{
	const uint32_t Output = GPIO_MODE_OUTPUT_PP;
	const uint32_t Speed = GPIO_SPEED_FREQ_LOW;
	const uint32_t Pull = GPIO_NOPULL;

	/* Configure output type */
	uint32_t OutputMode = GPIOC->MODER;
	OutputMode &= ~(GPIO_MODER_MODER0 << (0x2 * PinIndex));
	OutputMode |= (Output & 0x03) << (0x2 * PinIndex);
	GPIOC->MODER = OutputMode;

	/* Configure i/o output type  */
	uint32_t TypeMode = GPIOC->OTYPER;
	TypeMode &= ~(GPIO_OTYPER_OT_0 << (0x2 * PinIndex));
	TypeMode |= (((GPIO_MODE_OUTPUT_PP & 0x10) >> 4U) << (0x2 * PinIndex));
	GPIOC->OTYPER = TypeMode;

	/* Configure i/o output speed */
	uint32_t SpeedMode = GPIOC->OSPEEDR;
	SpeedMode &= ~(GPIO_OSPEEDER_OSPEEDR0 << (0x2 * PinIndex));
	SpeedMode |= (Speed << (0x2 * PinIndex));
	GPIOC->OSPEEDR = SpeedMode;

	/* Setup pull-up or pull-down for this pin */
	uint32_t PullUpDownMode = GPIOC->PUPDR;
	PullUpDownMode &= ~(GPIO_PUPDR_PUPDR0 << (0x2 * PinIndex));
	PullUpDownMode |= ((Pull) << (0x2 * PinIndex));
	GPIOC->PUPDR = PullUpDownMode;
}
```

```
void InitGPIOCPinAlternate(uint32_t PinIndex)
{
	const uint32_t Output = GPIO_MODE_AF_PP;
	const uint32_t Speed = GPIO_SPEED_FREQ_LOW;
	const uint32_t Pull = GPIO_NOPULL;

	/* Configure output type */
	uint32_t OutputMode = GPIOC->MODER;
	OutputMode &= ~(GPIO_MODER_MODER0 << (0x2 * PinIndex));
	OutputMode |= (Output & 0x03) << (0x2 * PinIndex);
	GPIOC->MODER = OutputMode;

	/* Configure i/o output type  */
	uint32_t TypeMode = GPIOC->OTYPER;
	TypeMode &= ~(GPIO_OTYPER_OT_0 << (0x2 * PinIndex));
	TypeMode |= (((GPIO_MODE_OUTPUT_PP & 0x10) >> 4U) << (0x2 * PinIndex));
	GPIOC->OTYPER = TypeMode;

	/* Configure i/o output speed */
	uint32_t SpeedMode = GPIOC->OSPEEDR;
	SpeedMode &= ~(GPIO_OSPEEDER_OSPEEDR0 << (0x2 * PinIndex));
	SpeedMode |= (Speed << (0x2 * PinIndex));
	GPIOC->OSPEEDR = SpeedMode;

	/* Setup pull-up or pull-down for this pin */
	uint32_t PullUpDownMode = GPIOC->PUPDR;
	PullUpDownMode &= ~(GPIO_PUPDR_PUPDR0 << (0x2 * PinIndex));
	PullUpDownMode |= ((Pull) << (0x2 * PinIndex));
	GPIOC->PUPDR = PullUpDownMode;
}

void InitButtonGPIO(uint32_t PinIndex)
{
	const uint32_t Output = GPIO_MODE_INPUT;
	const uint32_t Speed = GPIO_SPEED_FREQ_LOW;

	uint32_t OutputMode = GPIOA->MODER;
	OutputMode &= ~(GPIO_MODER_MODER0 << (0x2 * PinIndex));
	OutputMode |= (Output & 0x03) << (0x2 * PinIndex);
	GPIOA->MODER = OutputMode;

	  uint32_t TypeMode = GPIOA->OTYPER;
	  TypeMode &= ~(GPIO_OTYPER_OT_0 << (0x2 * PinIndex));
	  TypeMode |= (((GPIO_MODE_OUTPUT_PP & 0x10) >> 4U) << (0x2 * PinIndex));
	  GPIOA->OTYPER = TypeMode;

	  uint32_t SpeedMode = GPIOA->OSPEEDR;
	  SpeedMode &= ~(GPIO_OSPEEDER_OSPEEDR0 << (0x2 * PinIndex));
	  SpeedMode |= (Speed << (0x2 * PinIndex));
	  GPIOA->OSPEEDR = SpeedMode;
}

void WriteCharRaw(USART_TypeDef *Def, char Cur)
{
	if (Cur == '\n')
	{
		WriteCharRaw(Def, '\r');
	}
```

```
	Def->TDR = Cur;
}

void WriteChar(USART_TypeDef *Def, char Cur)
{
	WriteCharRaw(Def, Cur);
	while ((Def->ISR & USART_ISR_TC) != USART_ISR_TC)
	{
		/* add time out here for a robust application */
	}
}

void FiniWrite()
{
	USART3->ICR |= USART_ICR_TCCF; /* Clear transfer complete flag */
}

void WriteString(USART_TypeDef *Def, const char *Str)
{
	for (uint16_t Index = 0;; Index++)
	{
		char Cur = Str[Index];
		if (Cur == 0x00)
		{
			break;
		}
		WriteChar(Def, Cur);
	}
	FiniWrite();
}

char RecvChar(USART_TypeDef *Def)
{
	for (;;)
	{
		if ((Def->ISR & USART_ISR_RXNE) == USART_ISR_RXNE)
		{
			return Def->RDR;
		}
	}
}

void PollLoop()
{
	for (;;)
	{
		char c = RecvChar(USART3);
		switch (c)
		{
			case 'r':
			{
				ToggleLEDPin(GPIO_PIN_6);
				break;
			}

			case 'g':
			{
				ToggleLEDPin(GPIO_PIN_9);
				break;
```

```
                }

                case 'b':
                {
                        ToggleLEDPin(GPIO_PIN_7);
                        break;
                }

                case 'o':
                {
                        ToggleLEDPin(GPIO_PIN_8);
                        break;
                }

                default:
                {
                        WriteString(USART3, "bad command: expected one of \'r g b
o\'\n");
                        break;
                }
        }
    }
}

void IntrLoop()
{
    for (;;)
    {
        if (!New)
        {
            continue;
        }

        char c = GetColor();
        char v = GetStatus();
        WipeColor();

        switch (c)
        {
            case 'r':
            {
                DisableLEDPin(GPIO_PIN_6);
                if (v)
                {
                    EnableLEDPin(GPIO_PIN_6);
                }
                break;
            }

            case 'g':
            {
                DisableLEDPin(GPIO_PIN_9);
                if (v)
                {
                    EnableLEDPin(GPIO_PIN_9);
                }
                break;
            }
```

```
                  case 'b':
                  {
                        DisableLEDPin(GPIO_PIN_7);
                        if (v)
                        {
                              EnableLEDPin(GPIO_PIN_7);
                        }
                        break;
                  }

                  case 'o':
                  {
                        DisableLEDPin(GPIO_PIN_8);
                        if (v)
                        {
                              EnableLEDPin(GPIO_PIN_8);
                        }
                        break;
                  }

                  default:
                  {
                        WriteString(USART3, "bad command: expected one of \'r0 g0
b0 o0 r1 g1 b1 o1\'\n");
                        break;
                  }
            }
      }
}

void EnableLEDPin(uint32_t PinNo)
{
      GPIOC->BSRR = PinNo;
}

void DisableLEDPin(uint32_t PinNo)
{
      const uint32_t UpperHalf = 16;
      GPIOC->BSRR = PinNo << UpperHalf;
}


void ToggleLEDPin(uint32_t PinNo)
{
      if ((GPIOC->ODR & PinNo) != 0X00u)
      {
            DisableLEDPin(PinNo);
      }
      else
      {
            EnableLEDPin(PinNo);
      }

}
/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
```

```
{
	HAL_Init();
	SystemClock_Config();

	RCC->AHBENR |= RCC_AHBENR_GPIOAEN | RCC_AHBENR_GPIOCEN;
	RCC->APB1ENR |= RCC_APB1ENR_USART3EN | RCC_APB2ENR_SYSCFGEN;

	InitButtonGPIO(0);
	InitGPIOCPinAlternate(4);
	InitGPIOCPinAlternate(5);

	GPIOC->AFR[0] |= (1 << 16) | (1 << 20) ;

	for (char Index = 6; Index <= 9; Index++)
	{
		InitLEDGPIOPin(Index);
	}

	EXTI->IMR = 0x01;
	EXTI->FTSR = 0x00;
	EXTI->RTSR = 0x01;
	NVIC_EnableIRQ(EXTI0_1_IRQn);
	NVIC_SetPriority(EXTI0_1_IRQn, 3);
	NVIC_SetPriority(SysTick_IRQn, 2);

	/* Get the right baud rate... */
	uint32_t DestBaud = 115200;
	uint32_t SrcClock = HAL_RCC_GetHCLKFreq();
	uint32_t BaudBRR = SrcClock / DestBaud;

	USART3->BRR = BaudBRR;
	USART3->CR3 = USART_CR3_CTSE | USART_CR3_RTSE;

	#define INTR

	#if defined(INTR)
	NVIC_EnableIRQ(USART3_4_IRQn);
	USART3->CR1 = USART_CR1_RXNEIE | USART_CR1_RE | USART_CR1_UE | USART_CR1_TE;
	IntrLoop();
	#else
	USART3->CR1 = USART_CR1_TE | USART_CR1_RE | USART_CR1_UE;
	PollLoop();
	#endif
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
```

```
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```