УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Проектирование вычислительных систем»

Лабораторная работа №2

Вариант 6

Студент

Белогаев Д. В.

Кузнецов М. А.

P34131

Преподаватель

Пинкевич В. Ю.

Санкт-Петербург, 2023 г.

Цели лабораторной работы

1. Изучить протокол передачи данных по интерфейсу UART.
2. Получить базовые знания об организации системы прерываний в микроконтроллерах на примере микроконтроллера STM32.
3. Изучить устройство и принципы работы контроллера интерфейса UART, получить навыки организации обмена данными по UART в режимах опроса и прерываний.

Задание лабораторной работы

Разработать и реализовать два варианта драйверов UART для стенда SDK-1.1M: с использованием и без использования прерываний. Драйверы, использующие прерывания, должны обеспечивать работу в «неблокирующем» режиме (возврат из функции происходит сразу же, без ожидания окончания приема/отправки), а также буферизацию данных для исключения случайной потери данных. В драйвере, не использующем прерывания, функция приема данных также должна быть «неблокирующей», то есть она не должна зависать до приема данных (которые могут никогда не поступить). При использовании режима «без прерываний» прерывания от соответствующего блока UART должны быть запрещены. Написать с использованием разработанных драйверов программу, которая выполняет определенную вариантом задачу. Для всех вариантов должно быть реализовано два режима работы программы: с использованием и без использования прерываний. Каждый принимаемый стендом символ должен отсылаться обратно, чтобы он был выведен в консоли (так называемое «эхо»). Каждое новое сообщение от стенда должно выводиться с новой строки. Если вариант предусматривает работу с командами, то на каждую команду должен выводиться ответ, определенный в задании или «ОК», если ответ не требуется. Если введена команда, которая не поддерживается, должно быть выведено сообщение об этом.

Вариант задания

Доработать программу «светофор», добавив возможность отключения кнопки и задание величины тайм-аута (период, в течение

которого горит красный). Должны обрабатываться следующие команды, посылаемые через UART:

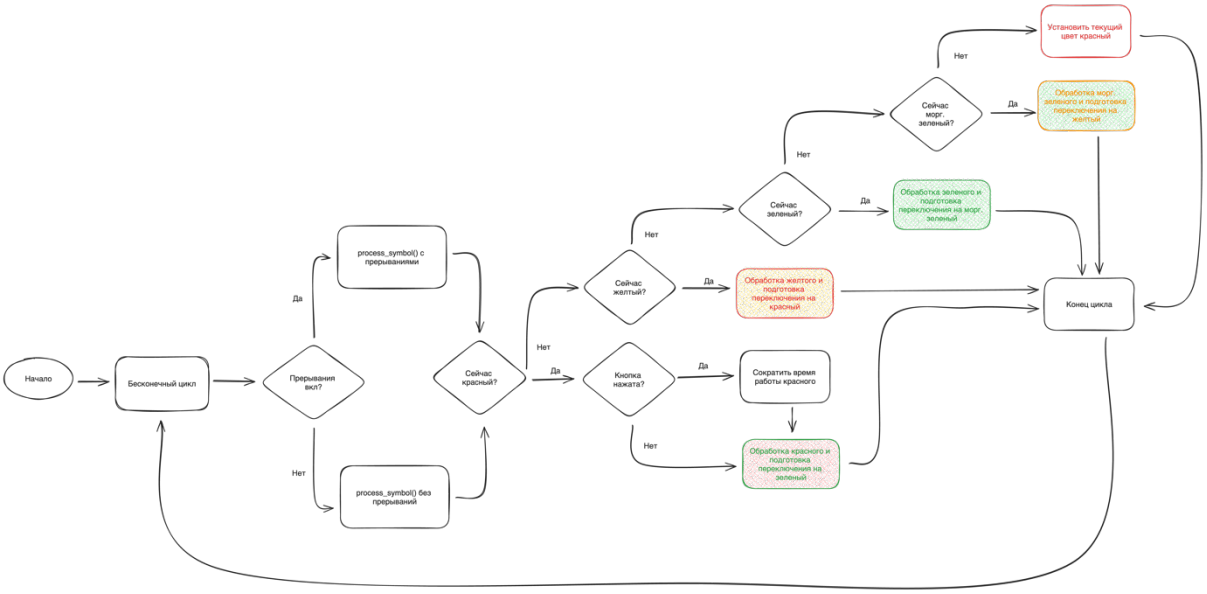
– ? – в ответ стенд должен прислать состояние, которое отображается в данный момент на светодиодах: red, yellow, green, blinking green, режим – mode 1 или mode 2 (см. далее), величину тайм-аута (сколько горит красный) – timeout …, и задействованы ли прерывания – символ I (interrupt) или P (polling);

– set mode 1 или set mode 2 – установить режим работы светофора, когда обрабатываются или игнорируются нажатия кнопки;

– set timeout X – установить тайм-аут (X – длина периода в секундах);

– set interrupts on или set interrupts off – включить или выключить прерывания.

Скорость обмена данными по UART: 57600 бит/с.



Исходный код

Объявления необходимых переменных:

```
uint16_t GREEN_LIGHT = GPIO_PIN_13;
uint16_t YELLOW_LIGHT = GPIO_PIN_14;
uint16_t RED_LIGHT = GPIO_PIN_15;
uint16_t BLINKING_GREEN = 0;
uint8_t INT_ON = 1;
```

```
uint8_t INT_OFF = 0;

uint16_t current_light = 1;
uint32_t start_time;
uint32_t duration = 1000;
uint32_t duration_for_red;
uint32_t duration_for_yellow = 1000;
uint32_t blink_duration = 100;
uint32_t blink_count = 0;
uint8_t button_flag = 0;

uint8_t interrupts_mode = 0;
uint8_t is_writing_now = 0;
char read_buffer[100];
char write_buffer[100];
uint8_t status;
char* cur_process_char = read_buffer;
char* cur_read_char = read_buffer;
char* transmit_from_pointer = write_buffer;
char* write_to_pointer = write_buffer;
uint8_t block_button = 0;
```

Функция ожидания заданного временного интервала:

```
void wait(uint32_t duration)
{
    uint32_t begin = HAL_GetTick();
    while((HAL_GetTick() - begin) < duration){}
}
```

Функция управления цветами светофора:

```
void turnSpecificLightOff(uint16_t light_type)
{
    HAL_GPIO_WritePin(GPIOD, light_type, GPIO_PIN_RESET);
}
```

```
void shutdownAll()
{
    turnSpecificLightOff(GREEN_LIGHT);
    turnSpecificLightOff(YELLOW_LIGHT);
    turnSpecificLightOff(RED_LIGHT);
}


void turnSpecificLightOn(uint16_t light_type)
{
    HAL_GPIO_WritePin(GPIOD, light_type, GPIO_PIN_SET);
}


void blinkLight(uint32_t count, uint16_t light_type, uint32_t
duration)
{
    for(uint32_t i = 0; i < count; i++)
    {
        wait(duration);
        turnSpecificLightOn(light_type);
        wait(duration);
        turnSpecificLightOff(light_type);
    }
}
```

Функции для работы со строками и обработки символов

```
char* concat(char *s1, char *s2) {
    char *result = malloc(strlen(s1) + strlen(s2) + 1);
    strcpy(result, s1);
    strcat(result, s2);
    return result;
}
```

```
void next_char(char **pointer, char *buffer) {
    if(*pointer >= buffer + 100){
        *pointer = buffer;
    }
    else {
        (*pointer)++;
    }
}

void write_char_to_buff(char c) {
    *write_to_pointer = c;
    next_char(&write_to_pointer, write_buffer);
}

void write(char* str) {
    char* str_with_newlines = concat("\r\n", str);
    str_with_newlines = concat(str_with_newlines, "\r\n");
    int size = sizeof(char)*strlen(str_with_newlines);
    if (interrupts_mode == 0) {
        HAL_UART_Transmit(&huart6, (uint8_t *) str_with_newlines,
size, 10);
    } else {
        for(size_t i = 0; str_with_newlines[i] != '\0'; i++) {
            write_char_to_buff(str_with_newlines[i]);
        }
    }
}

void write_not_found() {
    char* str = "Command not found. Available: '?', set mode 1/2,
set timeout X, set interrupts on/off";
    write(str);
}
```

```
int is_number(char* str) {
    for (size_t i = 0; str[i] != '\0'; i++) {
      if (!isdigit(str[i])) return 0;
    }
    return 1;
}

void process_symbol() {
    if (*cur_process_char == '\r') {
      *cur_process_char = '\0';
      char* command = strtok(read_buffer, " ");
      if (strcmp(command, "?") == 0) {
          char answer[100];
          char* light;
          switch (current_light) {
          case GPIO_PIN_15:
              light = "red";
              break;
          case GPIO_PIN_14:
              light = "yellow";
              break;
          case GPIO_PIN_13:
              light = "green";
              break;
          case 0:
              light = "blinking green";
              break;
          }
          uint8_t mode;
          char interrupts;
          if (block_button == 0) {
```

```
            mode = 1;
        } else mode = 2;
        if (interrupts_mode == 1) {
            interrupts = 'I';
        } else {
            interrupts = 'P';
        }
        sprintf(answer, "Light: %s, Mode: %d, Timeout: %d,
[I]nterrupts/[P]olling: %d", light, mode, duration*4,
interrupts_mode);
        write(answer);
    } else if (strcmp(command, "set") == 0) {
        char* first_arg = strtok(NULL, " ");
        if (strcmp(first_arg, "mode") == 0) {
            char* mode = strtok(NULL, " ");
            if (strcmp(mode, "1") == 0) {
                button_flag = 0;
                duration_for_red = duration * 4;
                block_button = 0;
                write("Entered mode 1");
            } else if (strcmp(mode, "2") == 0) {
                button_flag = 1;
                duration_for_red = duration;
                block_button = 1;
                write("Entered mode 2");
            } else {
                write_not_found();
            }
        } else if (strcmp(first_arg, "timeout") == 0) {
            char* timeout = strtok(NULL, " ");
            if (is_number(timeout)) {
                int new_dur = atoi(timeout) * 1000;
                if (duration_for_red == duration) {
```

```
                    duration_for_red = new_dur / 4;
                } else {
                    duration_for_red = new_dur;
                }
                write(concat("New duration is ", timeout));
                duration = new_dur / 4;
            } else {
                write_not_found();
            }
        } else if (strcmp(first_arg, "interrupts") == 0) {
            char* interrupts = strtok(NULL, " ");
            if (strcmp(interrupts, "on") == 0) {
                interrupts_mode = 1;
                transmit_from_pointer = write_to_pointer;
                cur_read_char = read_buffer;
                write("Interrupt mode on");
                HAL_UART_Receive_IT(&huart6, (uint8_t *)
cur_read_char, sizeof( char ));
            } else if (strcmp(interrupts, "off") == 0) {
                interrupts_mode = 0;
                HAL_UART_Abort_IT(&huart6);
                write("Interrupt mode off");
            } else {
                write_not_found();
            }
        } else {
            write_not_found();
        }
    } else {
        write_not_found();
    }
    cur_process_char = read_buffer;
```

```
  } else {
      next_char(&cur_process_char, read_buffer);
  }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    if (huart->Instance == huart6.Instance) {
        if (*cur_read_char == '\r'){
            cur_read_char = read_buffer;
        }
        else {
            write_char_to_buff(*cur_read_char);
            next_char(&cur_read_char, read_buffer);
        }
        if (interrupts_mode == 1)
            HAL_UART_Receive_IT(&huart6, (uint8_t *) cur_read_char,
sizeof( char ));
    }
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
    if (huart->Instance == huart6.Instance) {
        is_writing_now = 0;
        next_char(&transmit_from_pointer, write_buffer);
    }
}
```

Основная программа:

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
```

```
  /* MCU Configuration--------------------------------------------------------*/


  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();


  /* USER CODE BEGIN Init */


  /* USER CODE END Init */


  /* Configure the system clock */
  SystemClock_Config();


  /* USER CODE BEGIN SysInit */


  /* USER CODE END SysInit */


  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART6_UART_Init();
  /* USER CODE BEGIN 2 */
  duration_for_red = 4 * duration;
  if (interrupts_mode == 1) {
        HAL_UART_Receive_IT(&huart6, (uint8_t *) cur_read_char, sizeof( char ));
  }
  /* USER CODE END 2 */


  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
```

```
    {
      /* USER CODE END WHILE */


      /* USER CODE BEGIN 3 */
        switch(interrupts_mode) {
        case 1:
            if (cur_process_char != cur_read_char) {
                process_symbol();
            }
            if(is_writing_now == 0){
                if(transmit_from_pointer != write_to_pointer) {
                    is_writing_now = 1;
                    HAL_UART_Transmit_IT( &huart6, (uint8_t *)
transmit_from_pointer, sizeof( char ));
                }
            }
            break;
        case 0:
            status = HAL_UART_Receive(&huart6, (uint8_t *)
cur_process_char, sizeof( char ), 100);
            if (status == HAL_OK) {
                HAL_UART_Transmit(&huart6, (uint8_t *)
cur_process_char, sizeof( char ), 10);
                process_symbol();
            }
            break;
        }


        switch(current_light) {
        case GPIO_PIN_15:
            if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_15) == 0 &&
button_flag == 0 && block_button == 0) {
                duration_for_red = duration;
                button_flag = 1;
```

```
            }
            if ((HAL_GetTick() - start_time) >= duration_for_red) {
                current_light = GREEN_LIGHT;
                duration_for_red = 4 * duration;
                button_flag = 0;
                shutdownAll();
                turnSpecificLightOn(GREEN_LIGHT);
                start_time = HAL_GetTick();
            }
            break;
        case GPIO_PIN_14:
            if ((HAL_GetTick() - start_time) >= duration_for_yellow)
{
                current_light = RED_LIGHT;
                shutdownAll();
                turnSpecificLightOn(RED_LIGHT);
                start_time = HAL_GetTick();
            }
            break;
        case GPIO_PIN_13:
            if ((HAL_GetTick() - start_time) >= duration) {
                current_light = BLINKING_GREEN;
                shutdownAll();
                start_time = HAL_GetTick();
            }
            break;
        case 0:
            blinkLight(6, GREEN_LIGHT, blink_duration);
            current_light = YELLOW_LIGHT;
            turnSpecificLightOn(YELLOW_LIGHT);
            start_time = HAL_GetTick();
            break;
```

```
        default:
            current_light = RED_LIGHT;
            shutdownAll();
            turnSpecificLightOn(RED_LIGHT);
            start_time = HAL_GetTick();
        }
    }
  /* USER CODE END 3 */
}
```

Вывод

Во время выполнения лабораторной работы мы:

- изучили работу интерфейса передачи данных UART
- получили навыки организации обмена данными в двух режимах (Interrupts/Polling)