Университет ИТМО

Кафедра Вычислительной Техники

**Лабораторная работа №2**

**по дисциплине “Встроенные системы”**

Группа Р3401

Комаров Егор Андреевич

Вариант 1

Оценка:\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Принял: Ключев Аркадий Олегович

Санкт-Петербург, 2020

# Задание

Драйвер UART должен каждый принимаемый контроллером символ отсылать назад – так называемое «эхо». Каждое новое сообщение от стенда должно выводиться с новой строки.

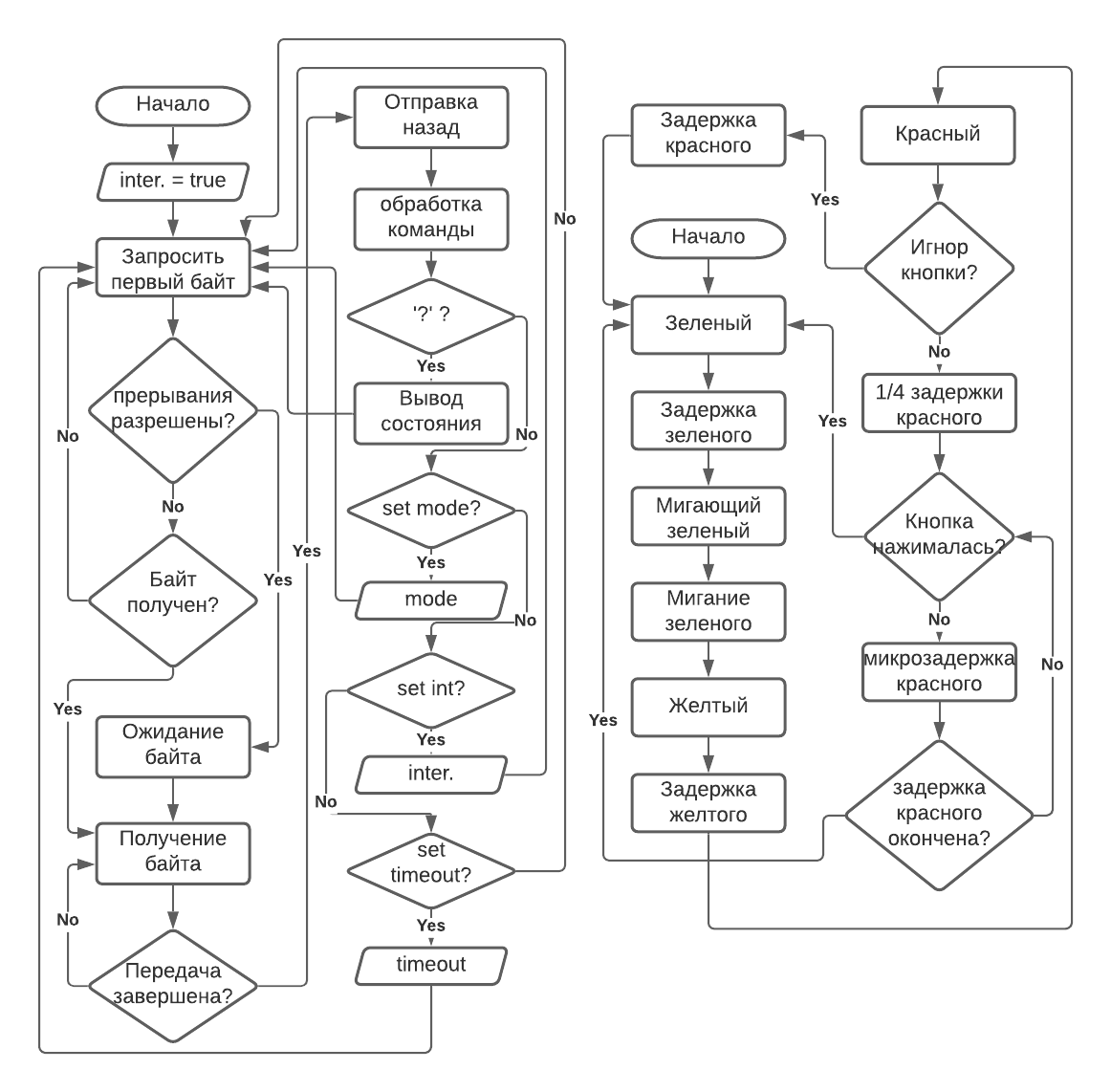
Связь между стендом и персональным компьютером (ПК) осуществляется с использованием терминальной программы – например, putty. Физически связь между ПК и стендом осуществляется через тот же кабель USB, через который происходит прошивка и отладка – со стороны SDK находится микросхема, которая управляется с помощью USB и формирует последовательные сигналы UART, которые подаются на входы контроллера. Реализуется два варианта программы: с использованием прерываний и без.

Доработать программу «светофор», добавив возможность отключения кнопки и задание величины таймаута (период, в течение которого горит красный).

Должны обрабатываться следующие команды, посылаемые через UART:

* *?* – в ответ контроллер должен прислать состояние, которое отображается в данный момент на диодах: *red*, *yellow*, *green*, *blinking green*, режим – *mode* 1 или *mode 2*, величину таймаута (сколько горит красный) – *timeout …*, и задействованы ли прерывания или нет – символ *I* или *P*;
* *set mode* 1 или *set mode 2* – установить режим работы светофора, когда обрабатываются или игнорируются нажатия кнопки;
* *set timeout X* – установить таймаут;
* *set interrupts on* или *set interrupts off* – включить или выключить прерывания в драйвере.

# Блок-схема



# Инструментарий

## Аппаратные средства

### Uart

В ходе работы инициализируется 2 интерфейса UART с целью подключения друг к другу, однако, из-за невозможности осуществить такую связь удаленно используется только один из них:

* UART2 на портах GPIO PA2 (TX) и PA3 (RX)
* UART3 на портах GPIO PB10 (TX) и PB11 (RX)

Характеристики:

* Асинхронный режим
* 115200 bps
* 8 бит сообщение
* Нет бита четности
* Стоп-бит – 1
* Receive и Transmit

## Программные средства

### Асинхронная передача через UART

Для асинхронной передачи через UART используются **HAL\_UART\_Receive\_IT** и **HAL\_UART\_Transmit\_IT**. При этом выполнение не блокируется, а при завершении операции возможно прерывание

### Синхронная передача через UART

Для синхронной передачи через UART используются **HAL\_UART\_Receive** и **HAL\_UART\_Transmit**. При этом выполнение блокируется до тех пор, пока не будет передано/принято указанное количество байт либо истечет timeout

### Использование прерываний UART

Для обработки прерываний использован callback **HAL\_UART\_RxCpltCallback** при завершении операции на прием (и **HAL\_UART\_TxCpltCallback** на передачу), переопределенный с пользовательским кодом

### Инициализация UART

Для инициализации использовалась функция **HAL\_UART\_Init,** принимающая структуру с параметрами UART (скорость, длина слова, параметры стоп битов, бит четности, режим чтения/записи и др)

### Подключение UART к GPIO

Для подключения UART к устройствам используются GPIO порты в режиме **GPIO\_MODE\_AF\_PP** и заданной альтернативной периферией в виде нужно UART

# Исходный код

## SDK

В качестве основы была взят код из прошлой лабораторной работы.

### Uart

В качестве API были добавлены функции для управления прерываниями и передачи данных известного размера. Приёмом управляет драйвер, который отсылает принятые данные обработчику команд.

#### SDK/interface.h

|  |
| --- |
| // uart  **#define** SDK\_UART\_HANDLE huart3  **#define** SDK\_UART\_TIMEOUT 3  **#define** SDK\_UART\_BUFFER\_SIZE 128  // uart API  **void** **SDK\_UART\_Init**();  **void** **SDK\_UART\_EnableInterrupts**(bool interrupts);  **void** **SDK\_UART\_Transmit**(uint8\_t\* pData, size\_t size);  bool **SDK\_UART\_IsInterruptible**(); |

Прием данных начинается с инициализации SDK\_UART. Сначала происходит ожидание получения первого байта в начало внутреннего буфера и затем побайтово записываются данные, пока не кончатся. После этого синхронно обрабатывается полученная команда и начинается новая итерация считывания.

#### SDK/uart.c

|  |
| --- |
| /// TYPES ///  **typedef** **struct** UartData  {  bool m\_interrupts;  uint8\_t m\_buffer[SDK\_UART\_BUFFER\_SIZE];  } UartData;  /// STATIC ///  **static** UartData s\_uartData;  **static** **void** **SDK\_UART\_StartReceiving**();  **static** **void** **SDK\_UART\_ContinueReceiving**();  **static void** **SDK\_UART\_Receive**(uint8\_t\* pData, size\_t size, size\_t offset);  **static** **void** **SDK\_UART\_StartReceiving**()  {  **if**(s\_uartData.m\_interrupts)  {  // will be recieved by callback  HAL\_UART\_Receive\_IT(&SDK\_UART\_HANDLE, s\_uartData.m\_buffer, 1);  }  **else**  {  // wait for first byte  **while**(HAL\_UART\_Receive(&SDK\_UART\_HANDLE, s\_uartData.m\_buffer, 1, SDK\_UART\_TIMEOUT) != *HAL\_OK*) { }  SDK\_UART\_ContinueReceiving(); }  }  **static** **void** **SDK\_UART\_ContinueReceiving**()  {  SDK\_UART\_Receive(s\_uartData.m\_buffer, SDK\_UART\_BUFFER\_SIZE, 1 );  SDK\_MAIN\_ProcessCommand((**const** **char**\*)s\_uartData.m\_buffer);  SDK\_UART\_StartReceiving();  }  /// API ///  **void** **SDK\_UART\_Init**()  {  s\_uartData.m\_interrupts = true;  SDK\_UART\_StartReceiving();  }  **void** **SDK\_UART\_EnableInterrupts**(bool interrupts)  {  s\_uartData.m\_interrupts = interrupts;  }  **void** **SDK\_UART\_Transmit**(uint8\_t\* pData, size\_t size)  {  HAL\_UART\_Transmit(&SDK\_UART\_HANDLE, pData, size, SDK\_UART\_TIMEOUT);  SDK\_DBG\_Print("Uart: %s", pData);  }  **void** **SDK\_UART\_Receive**(uint8\_t\* pData, size\_t size, size\_t offset)  {  uint32\_t dataInd = offset;  **while**(dataInd < size - 1 &&  HAL\_UART\_Receive(&SDK\_UART\_HANDLE, pData + dataInd, 1, SDK\_UART\_TIMEOUT) == *HAL\_OK*)  {  dataInd++;  }  pData[dataInd] = 0;  SDK\_UART\_Transmit(pData, dataInd);  }  bool **SDK\_UART\_IsInterruptible**()  {  **return** s\_uartData.m\_interrupts;  }  **void** **HAL\_UART\_RxCpltCallback**(UART\_HandleTypeDef \*huart)  {  SDK\_UART\_ContinueReceiving();  } |

### Timer

Для имитации приема команд аналогично нажатию кнопки из первой лабы были добавлены отложенные вызовы, но уже несколько штук и с аргументом. В остальном различий нет.

#### SDK/interface.h

|  |
| --- |
| // timer  **#define** SDK\_TIM\_INTERRUPT\_MAX 5  **#define** SDK\_TIM\_DATA\_INTERRUPT\_MAX 15  // timer API  **void** **SDK\_TIM\_Update**();  **void** **SDK\_TIM\_AddInterrupt**(**void**(\*callbackPtr)(), uint32\_t delay, bool periodic);  **void** **SDK\_TIM\_AddDataInterrupt**(**void**(\*callbackPtr)(**void**\* data), **void**\* data, uint32\_t delay, bool periodic); |

#### SDK/timer.c

|  |
| --- |
| **typedef** **struct** Interrupt  {  **void** (\*m\_callbackPtr)();  uint32\_t m\_delay;  uint32\_t m\_timer;  bool m\_periodic;  } Interrupt;  **typedef** **struct** DataInterrupt  {  **void** (\*m\_callbackPtr)(**void**\* data);  **void**\* m\_data;  uint32\_t m\_delay;  uint32\_t m\_timer;  bool m\_periodic;  } DataInterrupt;  /// STATIC ///  **static** Interrupt s\_interrupt[SDK\_TIM\_INTERRUPT\_MAX];  **static** uint32\_t s\_interruptCount = 0;  **static** DataInterrupt s\_dataInterrupt[SDK\_TIM\_DATA\_INTERRUPT\_MAX];  **static** uint32\_t s\_dataInterruptCount = 0;  /// API ///  **void** **SDK\_TIM\_AddInterrupt**(**void**(\*callbackPtr)(), uint32\_t delay, bool periodic)  {  **if**(s\_interruptCount < SDK\_TIM\_INTERRUPT\_MAX)  {  s\_interrupt[s\_interruptCount].m\_callbackPtr = callbackPtr;  s\_interrupt[s\_interruptCount].m\_delay = delay;  s\_interrupt[s\_interruptCount].m\_timer = 0;  s\_interrupt[s\_interruptCount].m\_periodic = periodic;  s\_interruptCount++;  }  }  **void** **SDK\_TIM\_AddDataInterrupt**(**void**(\*callbackPtr)(**void**\* data), **void**\* data, uint32\_t delay, bool periodic)  {  **if**(s\_dataInterruptCount < SDK\_TIM\_DATA\_INTERRUPT\_MAX)  {  s\_dataInterrupt[s\_dataInterruptCount].m\_callbackPtr = callbackPtr;  s\_dataInterrupt[s\_dataInterruptCount].m\_data = data;  s\_dataInterrupt[s\_dataInterruptCount].m\_delay = delay;  s\_dataInterrupt[s\_dataInterruptCount].m\_timer = 0;  s\_dataInterrupt[s\_dataInterruptCount].m\_periodic = periodic;  s\_dataInterruptCount++;  }  }  **void** **SDK\_TIM\_Update**()  {  **for**(**int** i = 0; i < s\_interruptCount; ++i)  {  **if**(s\_interrupt[i].m\_callbackPtr && ++s\_interrupt[i].m\_timer >= s\_interrupt[i].m\_delay)  {  s\_interrupt[i].m\_timer = 0;  s\_interrupt[i].m\_callbackPtr();  **if**(!s\_interrupt[i].m\_periodic)  {  s\_interrupt[i].m\_callbackPtr = 0;  }  }  }  **for**(**int** i = 0; i < s\_dataInterruptCount; ++i)  {  **if**(s\_dataInterrupt[i].m\_callbackPtr && ++s\_dataInterrupt[i].m\_timer >= s\_dataInterrupt[i].m\_delay)  {  s\_dataInterrupt[i].m\_timer = 0;  s\_dataInterrupt[i].m\_callbackPtr(s\_dataInterrupt[i].m\_data);  **if**(!s\_dataInterrupt[i].m\_periodic)  {  s\_dataInterrupt[i].m\_callbackPtr = 0;  }  }  }  } |

### Main

В главную обертку была добавлена функция для обработки данных, принятых из UART. Также требует определения в контексте приложения.

#### SDK/interface.h

|  |
| --- |
| **void** **SDK\_MAIN\_ProcessCommand**(**const** **char**\* command); |

## Application

### Semaphore

К светофору было добавлено несколько новых свойств согласно заданию.

#### App/semaphore.h

|  |
| --- |
| /// TYPES ///  **typedef** **enum** eColorState  {  *ECS\_Green*,  *ECS\_Yellow*,  *ECS\_Red*,  *ECS\_BlinkingGreen*  } eColorState;  **typedef** **enum** eSemaphoreMode  {  *ESM\_ProcessPress* = 1, // mode1  *ESM\_IgnorePress* = 2 // mode2  } eSemaphoreMode;  **typedef** **struct** SemaphoreState  {  eColorState m\_color;  eSemaphoreMode m\_mode;  uint32\_t m\_redTimeout;  } SemaphoreState;  SemaphoreState **SEM\_GetState**();  **void** **SEM\_SetMode**(eSemaphoreMode mode);  **void** **SEM\_SetRedTimeout**(uint32\_t timeout); |

#### App/semaphore.c

|  |
| --- |
| /// STATIC ///  **static** SemaphoreState s\_semaphoreState;  **static** uint16\_t **MapColorStateToLed**(eColorState color)  {  **switch**(color)  {  **case** *ECS\_Green*:  **return** SDK\_LED\_GREEN;  **case** *ECS\_Yellow*:  **return** SDK\_LED\_YELLOW;  **case** *ECS\_Red*:  **return** SDK\_LED\_RED;  **case** *ECS\_BlinkingGreen*:  **return** SDK\_LED\_GREEN;  }  **return** 0;  }  **static** eColorState **MapLedToColorState**(uint16\_t led, bool isBlinking)  {  **switch**(led)  {  **case** SDK\_LED\_GREEN:  **return** isBlinking ? *ECS\_BlinkingGreen* : *ECS\_Green*;  **case** SDK\_LED\_YELLOW:  **return** *ECS\_Yellow*;  **case** SDK\_LED\_RED:  **return** *ECS\_Red*;  }  **return** 0;  }  /// API ///  **void** **SEM\_Init**()  {  s\_semaphoreState.m\_color = *ECS\_Red*;  s\_semaphoreState.m\_mode = *ESM\_ProcessPress*;  s\_semaphoreState.m\_redTimeout = SEM\_RED\_PERIOD;  }  .....................  // red  SEM\_Show(SDK\_LED\_RED, s\_semaphoreState.m\_redTimeout, s\_semaphoreState.m\_mode == *ESM\_ProcessPress*);  .....................  SemaphoreState **SEM\_GetState**()  {  **return** s\_semaphoreState;  }  **void** **SEM\_SetMode**(eSemaphoreMode mode)  {  s\_semaphoreState.m\_mode = mode;  }  **void** **SEM\_SetRedTimeout**(uint32\_t timeout)  {  s\_semaphoreState.m\_redTimeout = timeout;  } |

### App cycle

В цикле приложения при старте откладывается несколько тестовых команд.

#### App/application.c

|  |
| --- |
| **void** **SDK\_MAIN\_PreLoop**()  {  SDK\_DBG\_Print("%s", "Begin simulation");  // init semaphore  SEM\_Init();  **#if** SDK\_REMOTE\_MODE  // simulate button press  SDK\_TIM\_AddInterrupt(&SDK\_BTN\_SetDown, SEM\_BTN\_PERIOD, true);  // simulate command input  SDK\_TIM\_AddDataInterrupt(&SDK\_MAIN\_ProcessCommand, "set interrupts 0", 100, false);  SDK\_TIM\_AddDataInterrupt(&SDK\_MAIN\_ProcessCommand, "set timeout 100", 100, false);  SDK\_TIM\_AddDataInterrupt(&SDK\_MAIN\_ProcessCommand, "set mode 2", 700, false);  SDK\_TIM\_AddDataInterrupt(&SDK\_MAIN\_ProcessCommand, "?", 1000, false);  **#endif**  }  **void** **SDK\_MAIN\_ProcessCommand**(**const** **char**\* command)  {  CMD\_ProcessCommand(command);  } |

### Commands

Модуль для обработки поступивших команд, перенаправленных через main\_wrapper из uart. В первую очередь вызывается функция **CMD\_ProcessCommand**, вызывающая **CMD\_ParseComand**. При успешном распознавании возвращаются вид и аргументы команды и только после происходит выполнение.

#### App/commands.h

|  |
| --- |
| /// TYPES ///  **typedef** **enum** eCmdType  {  *ECT\_GetInfo*, // '?'  *ECT\_SetMode*, // 'set mode X'  *ECT\_SetTimeout*, // 'set timeout X'  *ECT\_SetInterrupts*, // 'set interrupts X'  *ECT\_Undefined*  } eCmdType;  **typedef** **struct** CmdData  {  eCmdType m\_type;  uint32\_t m\_arg;  } CmdData;  /// API ///  **void** **CMD\_ProcessCommand**(**const** **char**\* command);  **void** **CMD\_GetInfo**();  CmdData **CMD\_ParseComand**(**const** **char**\* command);  bool **CMD\_ParseGetInfo**(**const** **char**\* command, **int** strSize, CmdData\* data);  bool **CMD\_ParseSet**(**const** **char**\* command, **int** strSize, CmdData\* data);  bool **CMD\_ParseSetMode**(**const** **char**\* command, **int** strSize, CmdData\* data);  bool **CMD\_ParseSetTimeout**(**const** **char**\* command, **int** strSize, CmdData\* data);  bool **CMD\_ParseSetInterrupts**(**const** **char**\* command, **int** strSize, CmdData\* data); |

#### App/commands.c

|  |
| --- |
| /// STATIC ///  **static** **char**\* **MapColorStateToName**(eColorState color)  {  **switch**(color)  {  **case** *ECS\_Green*:  **return** "Green";  **case** *ECS\_Yellow*:  **return** "Yellow";  **case** *ECS\_Red*:  **return** "Red";  **case** *ECS\_BlinkingGreen*:  **return** "Blinking Green";  }  **return** "";  }  **static** **char**\* **MapModeToName**(eSemaphoreMode mode)  {  **switch**(mode)  {  **case** *ESM\_ProcessPress*:  **return** "Process";  **case** *ESM\_IgnorePress*:  **return** "Ignore";  }  **return** "";  }  **static** **char**\* **MapBool**(bool value)  {  **if**(value)  {  **return** "true";  }  **return** "false";  }  /// API ///  **void** **CMD\_ProcessCommand**(**const** **char**\* command)  {  CmdData data = CMD\_ParseComand(command);  **switch**(data.m\_type)  {  **case** *ECT\_GetInfo*:  CMD\_GetInfo();  **break**;  **case** *ECT\_SetMode*:  SEM\_SetMode(data.m\_arg);  **break**;  **case** *ECT\_SetTimeout*:  SEM\_SetRedTimeout(data.m\_arg);  **break**;  **case** *ECT\_SetInterrupts*:  SDK\_UART\_EnableInterrupts(data.m\_arg);  **break**;  **default**:  **break**;  }  }  **void** **CMD\_GetInfo**()  {  SemaphoreState info = SEM\_GetState();  **char** buffer[128];  **sprintf**(buffer, "\nColor: %s \nModeOnPress: %s \nRed timeout: %d \nInterrupts: %s \n",  MapColorStateToName(info.m\_color), MapModeToName(info.m\_mode),  info.m\_redTimeout, MapBool(SDK\_UART\_IsInterruptible()));  SDK\_UART\_Transmit((uint8\_t\*)buffer, **strlen**(buffer));  }  // parsing  CmdData **CMD\_ParseComand**(**const** **char**\* command)  {  CmdData data;  data.m\_type = *ECT\_Undefined*;  data.m\_arg = 0;  size\_t strSize = **strlen**(command);  uint32\_t commandStart = 0;  // find first non-space symbol  **while** (commandStart < strSize && isspace((**int**)command[commandStart]))  {  commandStart++;  }  **if** (commandStart == strSize)  {  **return** data;  }  command = command + commandStart;  strSize -= commandStart;  // try parse as different commands  **if** (CMD\_ParseGetInfo(command, strSize, &data))  {  **return** data;  }  **if** (CMD\_ParseSet(command, strSize, &data))  {  **return** data;  }  **return** data;  }  bool **CMD\_ParseGetInfo**(**const** **char**\* command, **int** strSize, CmdData\* data)  {  **char**\* pCh = **strstr**(command, "?");  **if** (command && pCh == command &&  (strSize == 1 || isspace((**int**)command[1])))  {  data->m\_type = *ECT\_GetInfo*;  data->m\_arg = 0;  **return** true;  }  **return** false;  }  bool **CMD\_ParseSet**(**const** **char**\* command, **int** strSize, CmdData\* data)  {  **char**\* pCh = **strstr**(command, "set");  **if** (command && pCh == command &&  strSize > 3 && isspace((**int**)command[3]))  {  uint32\_t commandStart = 3;  **while** (commandStart < strSize && isspace((**int**)command[commandStart]))  {  commandStart++;  }  **if** (commandStart == strSize)  {  **return** false;  }  command = command + commandStart;  strSize -= commandStart;  **if** (CMD\_ParseSetMode(command, strSize, data))  {  **return** true;  }  **if** (CMD\_ParseSetTimeout(command, strSize, data))  {  **return** true;  }  **if** (CMD\_ParseSetInterrupts(command, strSize, data))  {  **return** true;  }  }  **return** false;  }  bool **CMD\_ParseSetMode**(**const** **char**\* command, **int** strSize, CmdData\* data)  {  **int** value;  uint32\_t success = **sscanf**(command, "mode %d", &value);  **if** (success == 1)  {  data->m\_type = *ECT\_SetMode*;  data->m\_arg = value;  **return** true;  }  **return** false;  }  bool **CMD\_ParseSetTimeout**(**const** **char**\* command, **int** strSize, CmdData\* data)  {  **int** value;  uint32\_t success = **sscanf**(command, "timeout %d", &value);  **if** (success == 1)  {  data->m\_type = *ECT\_SetTimeout*;  data->m\_arg = value;  **return** true;  }  **return** false;  }  bool **CMD\_ParseSetInterrupts**(**const** **char**\* command, **int** strSize, CmdData\* data)  {  **int** value;  uint32\_t success = **sscanf**(command, "interrupts %d", &value);  **if** (success == 1)  {  data->m\_type = *ECT\_SetInterrupts*;  data->m\_arg = value;  **return** true;  }  **return** false;  } |

## Driver

### Main

В функцию инициализации было добавлено создание двух UART

#### main.c

|  |
| --- |
| HAL\_Init();  SystemClock\_Config();  MX\_GPIO\_Init();  MX\_IWDG\_Init();  MX\_USART2\_UART\_Init();  MX\_USART3\_UART\_Init();  SDK\_MAIN\_Wrapper(); |

### UART

Для инициализации uart создается структура с характеристиками, описанными в инструментарии, и вызывается **HAL\_UART\_Init**

#### uart.c

|  |
| --- |
| **void** **MX\_USART2\_UART\_Init**(**void**)  {  huart2.Instance = USART2;  huart2.Init.BaudRate = 115200;  huart2.Init.WordLength = UART\_WORDLENGTH\_8B;  huart2.Init.StopBits = UART\_STOPBITS\_1;  huart2.Init.Parity = UART\_PARITY\_NONE;  huart2.Init.Mode = UART\_MODE\_TX\_RX;  huart2.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;  huart2.Init.OverSampling = UART\_OVERSAMPLING\_16;  **if** (HAL\_UART\_Init(&huart2) != *HAL\_OK*)  {  Error\_Handler();  }  }  /\* USART3 init function \*/  **void** **MX\_USART3\_UART\_Init**(**void**)  {  huart3.Instance = USART3;  huart3.Init.BaudRate = 115200;  huart3.Init.WordLength = UART\_WORDLENGTH\_8B;  huart3.Init.StopBits = UART\_STOPBITS\_1;  huart3.Init.Parity = UART\_PARITY\_NONE;  huart3.Init.Mode = UART\_MODE\_TX\_RX;  huart3.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;  huart3.Init.OverSampling = UART\_OVERSAMPLING\_16;  **if** (HAL\_UART\_Init(&huart3) != *HAL\_OK*)  {  Error\_Handler();  }  } |

# Выводы

Главной проблемой, приемлемого решения которой найти не удалось, оказалась невозможность отправки на UART каких-либо данных удаленно. В качестве примера на GitHub приведен как раз схожий случай, но на практике в данный момент нет соединенных друг с другом UART для возможности локальной посылки. В связи с этим поведение можно лишь примерно имитировать через отложенный ручной вызов команд.

Иных особых трудностей выполнение не вызывало. В процессе была освоены разные виды передачи через UART и обработка соответствующих прерываний.

Полный исходный код ЛР можно найти на github <https://github.com/Old-Fritz/EmbededSystems>

#### Результат работы:

