Университет ИТМО

Лабораторная работа № 3

по дисциплине «Встроенные системы»

Студент:
P3410 Нгу Фыонг Ань
Преподаватель:

Санкт-Петербург
2020 г.

Цель работы – освоение работы с последовательным интерфейсом микроконтроллера.

ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

1. Изучите схему последовательного интерфейса стенда.
2. Изучите принцип взаимодействия двух UART модулей SDK 1.1M на основе прерываний.
3. Изучите алгоритмы приема и передачи байта через последовательный канал.
4. Разработайте алгоритм программы, реализующей следующие действия:
    - отправку последовательности байт по UART;
    - прием последовательности байт, пересылаемых от одного UART модуля другому на основе прерываний;
    - обработку пересылаемых байт в обработчике прерывания;
5. Написать программное обеспечение обеспечивающее:
    a. инициализацию интерфейсов UART на стенде. Скорость передачи выбирается исходя из варианта;
    b. инициализацию обработчика прерываний интерфейсов UART;
    c. отправку двух uint8_t констант X и Y с первого UART модуля на второй*. Константы должны быть заданы любыми значениями от 0 до 255;
    d. вывод в трассировочный лог констант, переданных по интерфейсу;
    e. прием и обработку вторым UART модулем полученных значений в обработчике прерывания. Принятые константы должны быть также отображены в трассировочном логе. Результатом обработки должно являться uint8_t число, вычисленное на основе заданной функции по варианту (задание);
    f. отправку результата вычисленной функции со второго UART модуля на первый модуль;
    g. прием результата на первом UART модуле и вывод результата в трассировочный лог в формате, указанном в варианте задания;
6. Собрать проект в Debug режиме;
7. Загрузить сгенерированный бинарный файл на ITMO.cLAB для проверки правильности выполнения задания.
8. По итогам работы написать и защитить отчет. Отчет должен содержать описание теоретической и практической частей, а также основной код программы. В процессе защиты работы требуется демонстрация реализованного функционала, поэтому .bin файл для загрузки на ITMO.cLAB должен быть готов к началу сдачи.

№	Скорость передачи, bps	Задание	Формат вывода
2	19200	Z = X*Y	dec

1. Теория

1. UART-интерфейс

Универсальный асинхронный приемник / передатчик (UART) представляет собой блок схем, ответственный за реализацию последовательной связи. По сути, UART выступает в качестве посредника между параллельными и последовательными интерфейсами. На одном конце UART есть шина из восьми (или около того) линий данных (плюс некоторые управляющие контакты), с другой - два последовательных провода - RX и TX.

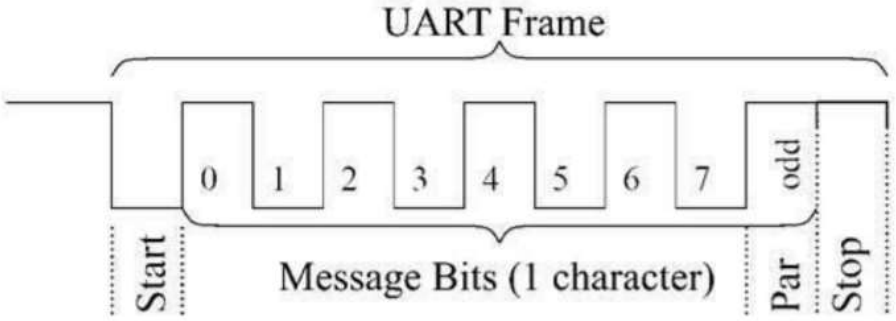

Рис. 14. Временная диаграмма передачи сообщения через UART

R и T в терминологии UART несут ответственность за отправку и получение последовательных данных. На стороне передачи UART должен создать пакет данных - добавление битов синхронизации и четности - и отправить этот пакет по линии TX в соответствии с установленной скоростью передачи. На стороне приема UART должен проверять линию RX со скоростью, соответствующей ожидаемой скорости передачи в бодах, выбирать биты синхронизации и выделять данные.

Поскольку сам по себе тактовый сигнал отсутствует, сначала отправляется «стартовый бит», чтобы сообщить получателю, что нужно прослушивать данные. Приемник наблюдает за переходом логического «высокого» в логический «низкий». Приемник синхронизирует свои собственные часы шины с этим битом.

После стартового бита идут биты, составляющие отправляемое «слово», причем первым отправляется нулевой бит, младший бит. Биты отправляются в виде импульсов по проводу через определенные интервалы времени, установленные на обоих концах связи на предварительно согласованные значения. Приемник в это время смотрит на напряжение на проводе; если он видит высокий логический уровень, он записывает двоичную цифру 1 или 0, если линия «низкий», или 0V. Приемник проверяет половину пути между началом и концом импульса, чтобы убедиться, что он не ошибочно считывает напряжение на линии в течение коротких интервалов, когда напряжение растет или падает.

Если два устройства согласились использовать «бит четности» для элементарной проверки ошибок, он вычисляется и отправляется затем синхронно с данными, которые были переданы на данный момент. Наконец, передатчик отправляет по крайней мере один «стоповый бит».

2. Последовательную передачу данных

Под последовательной передачей данных понимают процесс передачи данных по одному биту за один промежуток времени, последовательно один за одним по одному коммуникационному каналу или компьютерной шине, в отличие от параллельной передачи данных, при которой несколько бит пересылаются одновременно по линии связи из нескольких параллельных каналов.

Последовательная передача всегда используется при связи на дальние расстояния и в большинстве компьютерных сетей, так как стоимость кабеля и трудности синхронизации делают параллельную передачу данных неэффективной. При передаче данных на короткие расстояния последовательные компьютерные шины также используются всё чаще, так как и здесь недостатки параллельных шин перевешивают их преимущества в простоте. Развитие технологии для обеспечения целостности сигнала от передатчика до приёмника и достаточно высокая скорость передачи данных делают последовательные шины конкурентоспособными.

Возможны два режима последовательной передачи данных: синхронный и асинхронный.

Синхронный режим предполагает наличие средств синхронизации передатчика и приемника. Как правило, для синхронизации используют специальную линию для передачи тактовых импульсов. Информация в канале данных считывается приемником только в те моменты, когда на линии синхронизации сигнал активный.

В асинхронном режиме посылке очередного байта информации предшествует специальный старт-бит, сигнализирующий о начале передачи (обычно логический «0»). Затем следуют биты данных (их обычно 8), за которыми может следовать дополнительный бит (его наличие зависит от режима передачи, обычно этот бит выполняет функцию контроля четности). Завершается посылка стоп-битом (логическая «1»), длина которого (длительность единичного состояния линии) может соответствовать длительности передачи 1, 1.5 («полтора стоп-бита») или 2 бит. Стоп-бит гарантирует некоторую выдержку между соседними посылками, при этом пауза между ними может быть сколь угодно долгой (без учета понятия «тайм-аута»). Для асинхронного режима предусмотрен ряд стандартных скоростей обмена: 50, 75, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600 и 115200 bps.

3. Работу с последовательным интерфейсом в STM32CubeMX и в STM32CubeIDE с использованием фреймворка HAL

В библиотеке HAL существует две стандартных функции, отвечающих за прием и передачу информации:

- **HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout);** : Эта функция блокирующая, то есть пока она не выполнится программа будет приостановлена.
- **HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)**: Эта функция тоже блокирующая, она будет ожидать до тех пор пока не получит все запрошенные байты, или пока не истечет таймаут.

  • *huart – указатель на структуру UART_HandleTypeDef, которая содержит информацию о конфигурации для указанного модуля UART;

- *pData – указатель на буфер данных;
- Size – размер данных которые будут переданы или приняты;
- Timeout – длительность времени ожидания.

По окончании отправки сработает прерывание и вызовет колбек:
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)

Если для отправки данных блокирующая функция вполне подходит, то для приёма не годится. Поэтому нужно использовать функцию вызывающую прерывание.

HAL_UART_Transmit_IT (UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)
HAL_UART_Receive_IT (UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)

Эта функция не блокирует программу. Прерывание произойдет только после того, как всё запрошенные байты будут приняты.

Callback-функция завершения приема данных:
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)

2. Исходный код

Uart BaudRate config:

```
      void MX_USART2_UART_Init(void)
      {

        huart2.Instance = USART2;
        huart2.Init.BaudRate = 19200;
      //…
}
      void MX_USART3_UART_Init(void)
      {

        huart3.Instance = USART3;
        huart3.Init.BaudRate = 19200;
      //…
}

Main function:
      int main(void)
      {
        //init part

        uint8_t value_x = 4; // set X
        uint8_t value_y = 8; // set Y
        // buffer for UART2 transmit
        transmitBuffer2[0] = value_x;
        transmitBuffer2[1] = value_y;
        // buffer for UART3 receive
        receiveBuffer3[0] = 0;
        receiveBuffer3[1] = 0;
```

```
  // buffer for UART2 receive
  receiveBuffer2[0] = 0;

  SDK_TRACE_Timestamp(P0, 1);

  SDK_TRACE_Print("Value X: %d Value Y: %d", value_x, value_y);
  // Interrupt activations
  HAL_UART_Receive_IT(&huart3, receiveBuffer3, 2);
  HAL_UART_Transmit_IT(&huart2, transmitBuffer2, 2);
  HAL_Delay(100);

  //…
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
        if(huart == &huart3)
        {
             huart_Transfer(&huart3);
        } else if (huart == &huart2)
        {
             huart_Transfer(&huart2);
        }
}

void huart_Transfer(UART_HandleTypeDef *huart)
{
      if (huart == &huart3)
      {
            readed_value_x = receiveBuffer3[0];
            readed_value_y = receiveBuffer3[1];
            result = readed_value_x * readed_value_y;
            SDK_TRACE_Print("(UART3) X : %d", readed_value_x);
            SDK_TRACE_Print("(UART3) Y : %d", readed_value_y);
            SDK_TRACE_Print("(UART3) X*Y : %d", result);
            transmitBuffer3[0] = result;
            // our data is ready so we allow the reverse transfer
            HAL_UART_Receive_IT(&huart2, receiveBuffer2, 1);
            HAL_UART_Transmit_IT(&huart3, transmitBuffer3, 1);
      } else if (huart == &huart2)
      {
            readed_result = receiveBuffer2[0];
            SDK_TRACE_Print("(UART2) X*Y : %d", readed_result);
      }
}
```

3. Результат

Terminal

```
/****** SDK-1.1M Trace start (12/13/2020 4:52:31 AM) ******/
 0.018:    Value X: 4 Value Y: 8
 1.609:    (UART3) X : 4
 1.624:    (UART3) Y : 8
 1.639:    (UART3) X*Y : 32
 2.201:    (UART2) X*Y : 32
 /***************** SDK-1.1M Trace stop ******************/
```

4. Вывод

В ходе выполнения лабораторной работы была изучена последовательная передача данных, интерфейс USART и написана программа, реализующая взаимодействие двух UART модулей.