Практическая работа Тема 6. Асинхронный приёмо-передатчик

Цель работы: изучить принцип работы последовательного интерфейса UART, изучить порядок настройки и работы с интерфейсом на микроконтроллере Atmega328P.

Используемое оборудование:

Плата с МК Atmel AVR ATMega 328P	1 шт.
Блок питания 5В /3А	1 шт.
Программатор USBISP	1 шт.
USB кабель	1 шт.
Логический анализатор	1 шт.
Кнопочный модуль	1 шт.

Используемое ПО: Интегрированная среда разработки Atmel Studio 7.0 (или AVR Studio 4.19). Программа для загрузки программного кода в микроконтроллер AVRDUDEPROG

Теоретическая часть

UART (Universal Asynchroanous Receiver/Transmitter) – последовательный асинхронный интерфейс передачи данных. Для передачи используются два проводника RX (Receiver - приемник) и TX (Transmitter – передатчик), которые отвечают за передачу информации в противоположных направлениях. Обычно в передаче также используется линия GND. Схема соединения двух устройств по UART представлена на рисунке 6.1.

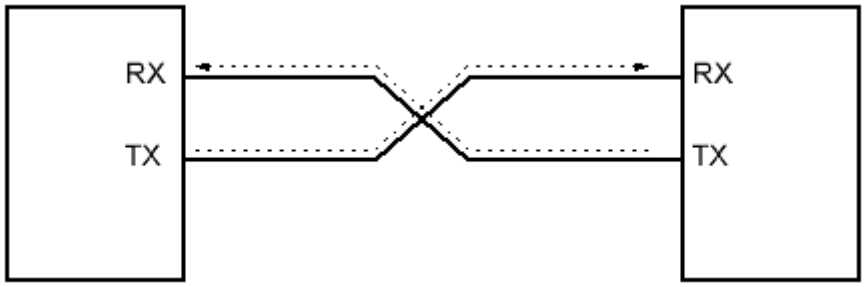

Рисунок 6.1 – Подключение двух устройств

Передача данных в UART выполняется следующим образом. В момент, когда линия свободна, на ней установлена 1. Первый бит пакета – стартовый, равен 0. Далее передаётся информационная часть, обычно вначале идут младшие биты. Информационная часть может содержать от 5 до 9 бит (в некоторых случаях даже до 12). После, если потребуется, передаётся бит чётности для контроля целостности данных (Если признак чётности вычисляемый приёмником не совпадет с передаваемым битом паритета, приёмник будет понимать, что данные переданы не корректно). Передача завершается стоповыми битами, которых может быть один или два. Протокол передачи данных по UART представлен на рисунке 6.2.

Примечание: На рисунке в скобках обозначены необязательные биты, их использование указывается в настройках работы интерфейса. Рисунок показывает особенности передачи, реализованные в МК AVR.

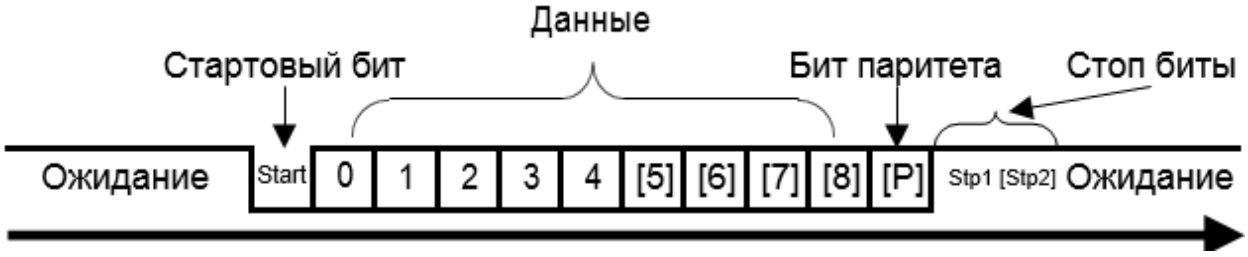

Рисунок 6.2 – Передача данных в UART

Скорость передачи определяется количеством бит, передаваемых в секунду (Количество бит в секунду также называют бодами). Существует общепринятый ряд стандартных скоростей: 300; 600; 1200; 2400; 4800; 9600; 19200; 38400; 57600; 115200; 230400; 460800; 921600 бит/с.

Для управления потоком данных UART используется программный или аппаратный метод.

В случае **программного метода**, информация о готовности устройства принимать данные или о необходимости остановить передачу передаётся по тем же каналам, что и данные. Принимающая сторона программно разделяет данные и управляющие сигналы в соответствии с принятым протоколом.

Аппаратное управление может использоваться некоторыми медленными устройствами или устройствами с простой схемной реализацией, однако оно потребует двух дополнительных линий для подключения устройства. При использовании аппаратного метода интерфейс UART предусматривает возможность использования дополнительных сигналов **CTS** и **RTS.** Подключение устройств с использованием аппаратного управления передачей изображено на рисунке 6.3.

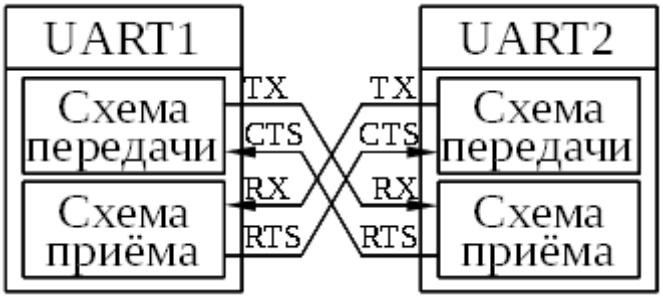

Рисунок 6.3 – Аппаратное управление

CTS (Clear To Send) – устанавливает принимающая сторона при готовности к приёму данных, активный уровень 0.

RTS (Request To Send) – запрос на отправку данных от передающей стороны для управления потоком данных.

Перед отправкой данных передатчик устанавливает сигнал **RTS** в 0. Если на **CTS** будет также установлен низкий уровень, передача происходит, иначе - нет. Если сигнал **CTS** будет установлен во время передачи информации, текущая передача всё равно будет завершена перед остановкой.

UART в AVR Atmel 328P

Практически для всех методов отладки программы необходимо физическое соединение с компьютером. Во многих платах Arduino имеется USB-разъём и специальная микросхема, которая преобразует интерфейс UART в USB. Построена такая микросхема на базе чипа CH340, ATmega, или др. Шина USB этих чипов подключена к порту USB, а шина UART к выводам TX и RX микроконтроллера.

Контроллер Atmega328P имеет один порт UART, сигналы которого подключены к выводам 0 (PORTD0 – сигнал RX) и 1 (PORTD1 – сигнал TX). Через эти выводы можно подключить к плате другое устройство имеющее интерфейс UART.

Также UART может работать и в синхронном режиме – USART, в этом случае будет задействована еще одна линия, которая будет обеспечивать синхронизацию передачи данных PortD4 XCK.

Примечание: в документации модуль UART в МК AVR обозначен как USART из-за наличия синхронного режима передачи.

UDR0 (UART Data Register) представляет собой два разных регистра, имеющих один адрес. Регистр буфера передачи данных (TXB) служит для записи данных, для приема данных используется буфер данных приема (RXB). При этом обращение и для записи и для чтения выполняется через константу UDR0.

7	6	5	4	3	2	1	0
RXB07	RXB06	RXB05	RXB04	RXB03	RXB02	RXB01	RXB00
TXB07	TXB06	TXB05	TXB04	TXB03	TXB02	TXB01	TXB00

Поскольку передача данных идет довольно медленно, то помещать данные в регистр UDR нужно только после окончания передачи предыдущего байта. О том, что UDR пуст и готов к приему нового байта, говорит бит **UDRE**, он же вызывает аппаратное прерывание по опустошению буфера.

В модулях UART буфер приемника является двухуровневым (FIFO-буфер), изменение состояния которого происходит при любом обращении к регистру UDR. В связи с этим не следует использовать регистр UDR в качестве операндов команд типа «чтение/модификация/запись» (SBI и CBI). Кроме того, следует быть очень аккуратными при использовании команд проверки `SBIC` и `SBIS`, т.к. они также изменяют состояние буфера приемника.

Примечание: для 5-, 6- или 7-битных пакетов старшая часть слова будет игнорироваться передатчиком и устанавливаться в ноль в получателе.

Регистры управления

UCSR0A – (UART Control Status Register) регистр состояния

7	6	5	4	3	2	1	0
RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0
R	R/W	R	R	R	R	R/W	R/W
0	0	1	0	0	0	0	0

Бит 7 RXC0 (Receive Complete) – флаг окончания приема данных: 1 – есть непрочитанные данные, 0 – регистр опустошен (только чтение).

Бит 6 TXC0 (Transmit Complete) – флаг окончания передачи данных: 1 – передача завершена, и в UDR0 не было загружено нового значения, 0 – выполняется передача.

Бит 5 UDRE0 (Data Register Empty) – флаг, означающий готовность регистра UDR получать новые данные. Если бит равен 1, то регистр UDR пуст и готов к приему новых данных (только чтение, после перезагрузки установлен в 1).

Бит 4 FE0 (Frame Error) – флаг ошибки кадрирования (фрейма). При обнаружении ошибки кадрирования (первый стоп-бит равен 0) устанавливается в 1, сбрасывается в 0 при приеме стоп-бита, равного 1 (только чтение).

Бит 3 DOR0 (Data OverRun) – флаг переполнения регистра данных: 1 – если в момент обнаружения нового старт-бита в сдвиговом регистре находится последнее принятое слово, а буфер приемника полон (только чтение).

Бит 2 UPE0 (Parity Error) – флаг ошибки четности: 1 – при ошибке четности (только чтение).

Бит 1 U2X0 – удвоение скорости обмена, если бит равен 1 (только в асинхронном режиме. В синхронном следует установить этот бит в 0).

Бит 0 MPCM0 – мультипроцессорный режим коммуникации. Если установлен в 1, режим включен.

UCSR0B – регистр управления

7	6	5	4	3	2	1	0
RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80
R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
0	0	0	0	0	0	0	0

Бит 7 RXCIE0 (RX Complete Interrupt Enable) – разрешение прерывания при завершении приема**,** если установлен в 1.

Бит 6 TXCIE0 (TX Complete Interrupt Enable) – разрешение прерывания при завершении передачи, если установлен в 1.

Бит 5 UDRIE0 (USART Data Register Empty Interrupt Enable) – разрешение прерывания при очистке регистра данных, если установлен в 1.

Бит 4 RXEN0 (Receiver Enable) – разрешение приема данных, если установлен в 1.

Бит 3 TXEN0 (Transmitter Enable) – разрешение передачи данных, если

установлен в 1.

Бит 2 UCSZ02 (Character Size) – формат посылки данных (используется совместно с битами UCSZ01 и UCSZ00 регистра UCSR0C (см. табл. 8.3).

Бит 1 RXB80 (Receive Data Bit 8) – 8-й разряд принимаемых данных при использовании 9-разрядных слов.

Бит 0 TXB80 (Transmit Data Bit 8) – 8-й разряд передаваемых данных при использовании 9-разрядных слов.

UCSR0C – регистр управления. Отвечает за режим работы UART, систему управления и длину передаваемых пакетов.

7	6	5	4	3	2	1	0
UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	1	1	0

Биты 7-6 UMSEL01 - UMSEL00 определяют режим работы контроллера UART (таблице 6.1).

Таблица 6.1 – Режимы работы UART

UMSEL01	**UMSEL00**	**Режим работы**
0	0	Асинхронный режим
0	1	Синхронный режим
1	0	Резерв
1	1	Ведущий SPI

Биты 5-4 UMP01 - UMP00 – режим проверки четности (таблица 6.2).

Таблица 6.2 – Система контроля и формирование четности в UART

UMP01	**UMP00**	**Режим работы системы контроля и формирования четности USART0**
0	0	Выключена система контроля
0	1	Резерв
1	0	Проверка четности
1	1	Проверка нечетности

Бит 3 USBS0 устанавливает количество стоп-битов (1 стоп-бит, если сброшен в 0. 2 стоп-бита, если установлен в 1).

Биты 2-1 UCSZ01 - UCSZ00 определяют количество бит данных в передаваемом пакете (совместно с битом UCSZ02 регистра UCSR0B). По умолчанию установлены две единицы – режим 8 бит.

Таблица 6.3 – Количество бит данных в пакете

UCSR0B	**UCSR0C**		**Количество бит данных в пакете**
UCSZ02	**UCSZ01**	**UCSZ00**	
0	0	0	5 бит
0	0	1	6 бит
0	1	0	7 бит
0	1	1	8 бит
1	0	0	Резерв
1	0	1	Резерв
1	1	0	Резерв
1	1	1	9 бит

Бит 0 UCPOL0 устанавливает полярность тактовых сигналов: 0 – передача по спаду, прием по нарастанию; 1 – передача по нарастанию, прием по спаду (для режима USART).

Регистр UBRR0 (Baud Rate Register)

Регистр UBRR0 определяет скорость передачи данных. UBRR0 является 12-разрядным регистром, состоящим из двух частей: UBBR0H (UART Baud Rate Register Hight), где хранятся 4 старших бита, и UBBR0L (UART Baud Rate Register Low), где хранятся 8 младших бит. В UBRR0 записывается определенное значение в зависимости от тактовой частоты и скорости передачи. Значение вычисляется по формуле:

$$\mathrm{BAUD} = \frac{\mathrm{CLK}}{16 \cdot (\mathrm{UBRR0} + 1)} \quad (1),$$

где **CLK** – тактовая частота микроконтроллера, а **BAUD** – требуемая частота в бодах, **UBRR0** – значение регистра.

Пример: предположим, что требуется установить скорость передачи, равную 2400 бит/с (бод)

Выразим из (1) UBRR0. Получаем формулу (2).

$$\mathrm{UBRR0} = \frac{\mathrm{CLK}}{16 \cdot \mathrm{BAUD}} - 1 \quad (2)$$

BAUD = 2400 бит/с, CLK = 16 * 10^6 (частота процессора 16 МГц). Подставляем эти значения в формулу 2. Получаем UBRR0 = $416_{<10>}$.

Значения для регистра UBRR просчитаны в технической документации на контроллер, и для тактового генератора 8МГц, который используется на отладочной плате, представлены в таблице 6.4.

Таблица 6.4. Значения UBRR для различных частот передачи
(из официальной технической документации)

Скорость передачи (бит/с)	**Частота генератора f_{osc}=16000000Гц**			
	U2Xn=0		**U2Xn=1**	
	UBRRn	**ОШИБКА**	**UBRRn**	**ОШИБКА**
2400	416	-0.1%	832	0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14400	68	0.6%	138	-0.1%
19200	51	0.2%	103	0.2%
28800	34	-0.8%	68	0.6%
38400	25	0.2%	51	0.2%
57600	16	2.1%	34	-0.8%
76800	12	0.2%	25	0.2%
115200	8	-3.5%	16	2.1%
230400	3	8.5%	8	-3.5%
250000	3	0%	7	0%
500000	1	0%	3	0%
1000000	0	0%	1	0%

Сводная таблица описанных регистров представлена в таблица 8.5.

Таблица 8.5 –Сводная таблица регистров UART

Адрес в ОЗУ	Адрес в I/O Reg	Имя регистра	Назначение битов регистра [7:0]							
			7	6	5	4	3	2	1	0
0xC0		UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0
0xC1		UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80
0xC2		UCSR0C	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0
0xC4		UBRR0L	UBRR[7:0]							
0xC5		UBRR0H					UBRR[8:11]			
0xC6		UDR0	TXB[7:0]/RXB[7:0]							

Все перечисленные в таблице регистры лежат в сегменте «Дополнительных регистров ввода/вывода» (Extended IN/OUT registers), расположенного с адреса 0x60 до адреса 0xFF. Обращение к этим регистрам на языке Assembler выполняется командами `STS, LDS`.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Монитор порта на ARDUINO IDE

Для обмена данными необходимы два устройства. Первое – отладочная плата микроконтроллера. В качестве второго будет использоваться персональный компьютер с программой Arduino IDE.

ARDUINO IDE - это среда разработки программного обеспечения, использующая C++ и предназначенная для программирования плат Arduino.

После подключения платы через USB необходимо определить, какое имя оно получило в системе. Для этого в диспетчере устройств (правая кнопка мыши по значку «Мой компьютер»/диспетчер устройств/Порты Com/LPT).

Далее необходимо открыть ARDUINO IDE, выбрать во вкладке «Инструменты» плату Arduino Uno (см. рисунок 6.5).

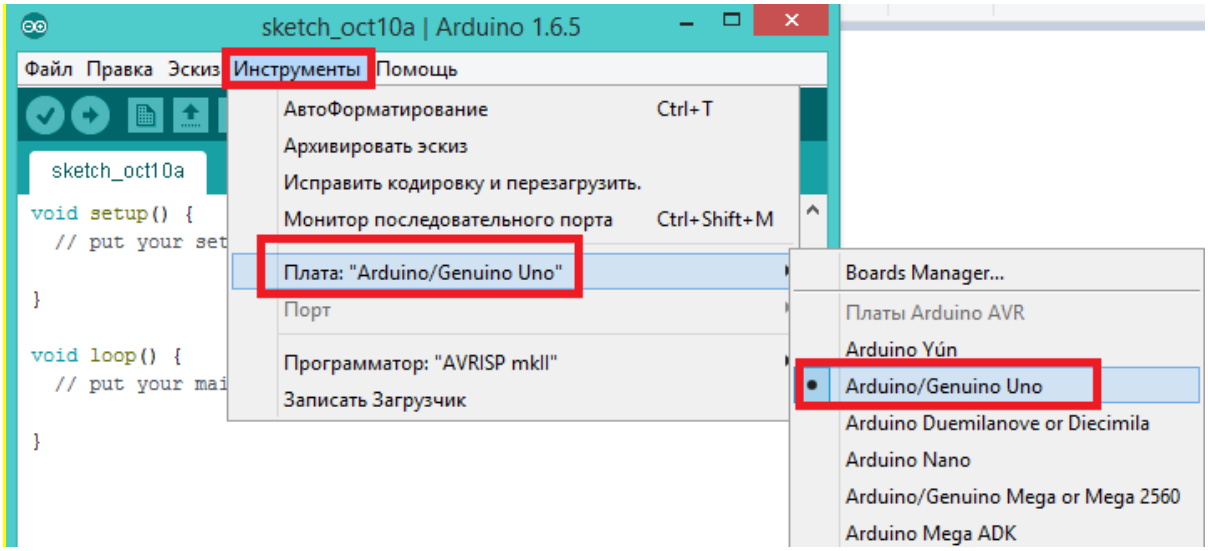

Рисунок 6.5 – Выбор платы в ARDUINO IDE

Далее необходимо выбрать в разделе соответствующий COM порт как на рисунке 6 (в примере COM4, имя порта может отличаться).

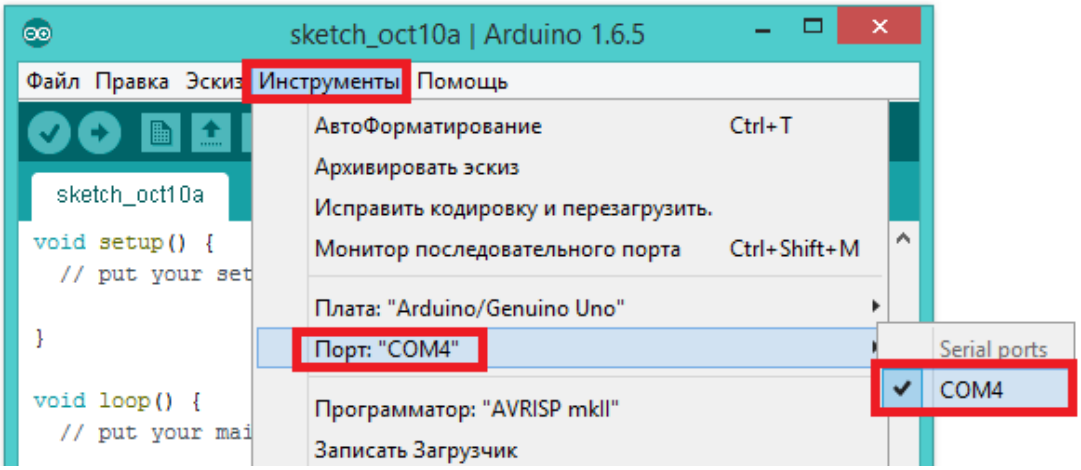

Рисунок 6.6 – Выбор COM порта

Затем вкладке «Инструменты» необходимо выбирать «Монитор последовательного порта». Теперь, нажимая на кнопку, подключенному ко 2 выводу PORTD будет выводится символ.

Инициализация и порядок использования UART

Перед началом работы с UART в МК Atmega328P необходимо:

1. Установить скорость обмена. В регистр UBBR0 установите значение из таблицы 6.4 (или вычислите по формуле 2), соответствующее необходимой скорости приема/передачи (для 9600 бит/с при частоте тактирования микроконтроллера 16 МГц – UBRR0 это значение будет равно 103, см. таблицу 6.4).

2. Выполнить настройку параметров обмена (использование бита четности, количество стоп-битов, передаваемый объем данных за один пакет, асинхронный режим). В рамках работы используем асинхронный обмен без бита четности, количество стоп битов – 1, в пакете 8 бит данных. Это соответствует параметрам по умолчанию. Но для обеспечения надежности следует установить 1 в биты UCSZ01 и UCSZ00 регистра UCSR0C, что включит режим передачи 8 бит. Остальные биты регистров UCSR0C и биты UCSR0B установите в 0.

3. Включить обработку необходимых прерываний. Обмениваться данными без использования системы прерываний очень трудно. Особенно учитывая, что частота тактирования контроллера 16 МГц (16000000 простейших операций в секунду), а скорость обмена 9600 бит/с. Один пакет по умолчанию состоит из 8 бит, стартового бита и стоп-бита, т.е. скорость передачи 960 пакетов в секунду. Эффективно и удобно использовать прерывания по приему и передаче пакета. Процессор микроконтроллера может выполнять свои задачи и, только когда завершается прием или передача, считывать полученные данные или посылать новые.

Чтобы включить обработку прерываний от UART, необходимо в регистре UCSR0B установить в единицу биты RXCIE0 (данные получены), TXCIE0 (данные переданы), UDRIE0 (регистр данных очищен), при этом флаг I регистра SREG (глобальное разрешение прерываний) должен быть установлен в 1. Данным прерываниям соответствуют векторы, описанные в таблица 6.6.

Таблица 6.6 – Векторы прерываний UART

Адрес вектора	Имя вектора	Событие вызывающее прерывание
0x0024	USART_RX	Прием завершен
0x0026	USART_UDRE	Регистр данных очищен
0x0028	USART_TX	Передача завершена

4. Включить режимы приема и передачи. Для включения приема установите 1 в бит RXEN0 регистра UCSR0B, для включения передачи – 1 в бит RXEN0 регистра UCSR0B.

Обеспечение приема/передачи данных:

1. **Для отправки данных** необходимо записать их в регистр UDR0, при этом учитывая, что передача занимает много времени и следующий байт можно отправлять только после полной передачи текущего. Для того чтобы убедиться, что данные переданы, требуется анализировать бит TXC0 (завершение передачи) или UDRE0 (регистр UDR0 - пуст) регистра UCSR0A, а более эффективно – не тратить процессорное время на циклический опрос регистра UCSR0A и проверку его битов, а использовать аналогичные прерывания (USART_RX или USART_UDRE – см. таблицу 6.6), и при их возникновении отправлять следующий байт.

2. **Для получения данных** необходимо считать регистр UDR0. Однако момент времени, в который будет получен новый байт, неизвестен и чтобы считать следующее слово требуется анализировать бит RXC0 регистра UCSR0A. Но как и в случае с передачей данных, чтобы не загружать процессор постоянной проверкой флага завершения приема, можно использовать соответствующее прерывание (USART_RX – см. таблицу 6.6), при возникновении которого можно считывать данные из регистра UDR0.

Обеспечение приема/передачи данных:

1. **Для отправки данных** запишите их в регистр UDR0, но необходимо учитывать, что передача занимает много времени, и следующий байт можно отправлять только после полной передачи текущего. Для этого выполняется анализ бита TXC0 (завершение передачи) или UDRE0 (регистр UDR0 пуст) регистра UCSR0A. Но более эффективно не тратить процессорное время на циклический опрос регистра UCSR0A и проверку его битов, а использовать аналогичные прерывания (USART_RX или USART_UDRE – см. таблицу 6.6), и при их возникновении отправлять следующий байт.

2. **Для получения данных** считайте регистр UDR0. Однако момент времени, в который будет получен новый байт, неизвестен, и для считывания следующего слова необходимо анализировать бит RXC0 регистра UCSR0A. Но, как и в случае с передачей данных, чтобы не загружать процессор постоянной проверкой флага завершения приема, используйте соответствующее прерывание (USART_RX – см. таблицу 6.6), при возникновении которого можно считывать данные из регистра UDR0.

Пример 1. Настройка интерфейса и передача одного байта данных.

Рассмотрим пример отправки байта информации при нажатии на кнопку. В данном примере не производится проверок опустошения буфера, но полностью описана инициализация параметров UART.

```
1  .include "m328Pdef.inc"
2  .org 0
3    jmp Reset
4  .org 0x002
5    jmp knopka
6  Reset:
7    ldi r16, 0x03
8    sts EICRA, r16
9    ldi r16, 0x01
10   out EIMSK, r16
11   cbi ddrd, 2
12   sbi portd,2
13 .equ CLK=16000000
14 .equ BAUD=9600
15 .equ UBRR0_value=(CLK/(BAUD*16))-1
16   ldi r16, high(UBRR0_value)
17   sts UBRR0H, r16
18   ldi r16, low(UBRR0_value)
19   sts UBRR0L, r16
20   ldi r16,(1<<TXEN0)
21   sts UCSR0B,R16
22   ldi r16,(1<<UCSZ00)|(1<< UCSZ01)
23   sts UCSR0C,R16
24   sei
25 main:
26   jmp main
27 knopka:
28   ldi r16, 0x3E
29   sts UDR0,r16
30 Reti
```

1 строка – подключение библиотеки для ATmega328P;

2-5 строки – инициализация векторов прерываний. В рамках этого примера используется прерывание INT0;

7-10 строки – настройка прерываний (см. тему №5 «обработка прерываний»);

11-12 строки – настройка вывода PORTD2 на прием информации и включении подтяжки;

13-20 строки – задание скорости передачи – бит/с (бод). *ВАЖНО: если скорости приемника и передатчика не будут совпадать, то передача данных будет некорректной*. 13 строка – тактовая частота генератора платы. 14 строка – требуемая скорость. 15 строка – вычисляем значение, которое нужно занести в UBBR0 (см. в описании лабораторной работы). Вместо строк 13-15 можно просто записать константу из таблицы 4. Так как регистр UBBR0L является 12-разрядным, то занесение числа должно проходить в 2 этапа – сначала заносим старшую часть, после – младшую (16-20 строки).

Примечание: заметим, что вычисления в строке 15 записаны не в виде ассемблерных команд, к тому же содержат операцию деления. Это не вызовет ошибку и не затруднит выполнение программы. Данное выражение – директива препроцессора среды Atmel Studio (AVR Studio) будет рассчитано еще на этапе компиляции программы и в МК будет записано конкретное значение.

20-23 строки – настройка параметров UART. 20 и 21 – настройка на передачу данных. 22 и 23 – установка количества бит данных в пакете;

24 строка – глобальное разрешение прерываний;

25-26 строки – основная программа (в примере не выполняет действий);

27-30 строки – подпрограмма прерывания INT0 – внешнее прерывание, возникновение которого определяет клавиша, подключенная к PortD2. Посылает символ «>» (в кодировке ASCII = 3E, табл. 6.7).

Табл. 6.7. Таблица кодировки ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	“	#	$	%	&		(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	O
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Выполните сборку программы. Подключите плату через USB кабель к компьютеру. К порту D2 (2 вывод платы) подключите кнопку. Проанализируйте работу контроллера. Загрузите код в контроллер. Прием информации на компьютере выполните с помощью монитора последовательного порта среды Arduino IDE. Результат зафиксируйте в виде снимка экрана «Монитор последовательного порта».

Ниже представлена аналогичная программа на языке С.

```
1  #include <avr/io.h>      //библиотеки ввода/вывода
2  #include <avr/interrupt.h> //библиотека обработки прерываний
3  ISR (INT0_vect)          //обработчик прерывания INT0
4  {
5     UDR0 = 0x3E;        //отправка символа ">" по UART
6  }
7  int main(void)           //точка входа в программу
8  {
9     EICRA |= 0x03;        //прерывание INT0 по фронту входящего сигнала
10    EIMSK |= 0x01;        //разрешение прерывания INT0
11    DDRD  &= ~0x04;       //PORTD2 - на ввод информации
12    PORTD |= 0x04;        //PORTD2 (INT0) - режим PULL UP
13    UBRR0 = 103;        //скорость 9600бит/с при частоте генератора 16МГц
14    UCSR0B |= (1<<TXEN0); //разрешение отправки данных по UART
15    UCSR0C |= (1<<UCSZ01)|(1<<UCSZ00);//размер пакета 8бит
16    sei();              //глобальное разрешение прерываний
17    while (1)           //бесконечный цикл (без действий)
18    {
19    }
}
```

Код программы на языке С значительно компактнее, чем на языке Assembler. Первые две строки отвечают за подключение библиотек системы ввода вывода и прерываний. В строках 3-6 описана подпрограмма обработчика прерываний – отправка сообщения с кодом 0x3E по UART. Все прерывания обрабатываются через процедуру ISR (<вектор>) в которой в качестве параметра указывается вектор прерывания.

В основной подпрограмме происходит инициализация работы порта:

- настройка прерываний от внешних источников;
- инициализация работы вывода порта D для захвата прерывания;
- в строке 13 установка скорости передачи информации (9600 бит/с при частоте синхронизации МК 16 МГц) через запись константы из таблицы 6.4;
- строка 14 – включение режима передачи – используемая структура (1<<TXEN0) – сдвиг единицы на количество разрядов, равное константе TXEN0, она хранит номер соответствующего бита в регистре USCR0B;
- строка 15 – установка режима приема/передачи – 8 бит.

Выполните сборку программы. Подключите оборудование согласно аналогично примеру на Assembler. Загрузите код в контроллер. Проанализируйте работу контроллера. Прием информации на компьютере выполните с помощью монитора последовательного порта среды Arduino IDE. Результат зафиксируйте в виде снимка экрана «Монитор последовательного порта».

Задание 1. Анализ передачи в логическом анализаторе.

Измените передаваемый символ в соответствии с таблицей 6.7. Выполните сборку программы. Подключите логический анализатор к выводу PORTD0. Загрузите код в контроллер. Откройте программу Logic. Выполните анализ сигнала с линии TX. Докажите, что сигнал на линии соответствует символу, передаваемому по варианту задания.

Таблица 6.7 – Варианты для задания 1

№ варианта	1	2	3	4	5	6	7	8	9	10
Символ	=	A	v	R	/	#	W	@	i	T

Пример 2. Передача сообщений.

В рассмотренных задачах выполняется передача по одному байту. Каждая передача выполняется по нажатию на клавишу. Чтобы контроллер мог без ошибок передать несколько байт сразу, необходимо точно убедиться, что предыдущая передача завершена. Для этого необходимо анализировать регистр UCSR0A (UDRE0 и TXC0).

Рассмотрим пример передачи сообщения, записанного в ПЗУ микроконтроллера на языке Assembler. Программа постоянно выводит через UART одно и тоже сообщение.

```
1  .include "m328Pdef.inc"
2  .equ CLK=16000000
3  .equ BAUD=9600
4  .equ UBRR0_value = (CLK/(BAUD*16)) - 1
5  .org 0
6     jmp Reset
7  Reset:
8  ;настройка UART
9  ;установка частоты приёма/передачи
10    ldi r16, high(UBRR0_value)
11    sts UBRR0H, r16
12    ldi r16, low(UBRR0_value)
13    sts UBRR0L, r16
14 ;разрешение передачи
15    ldi r16,(1<<TXEN0)
16    sts UCSR0B,R16
17 ;длина слова 8 бит
18    ldi r16,(1<< UCSZ00)|(1<< UCSZ01)
19    sts UCSR0C,R16
20 ;разрешение всех прерываний
21    sei
22 main:
23 ;определение адреса сообщения в ПЗУ
24    ldi zl,low(Msg<<1)
25    ldi zh,high(Msg<<1)
26 ;количество байт в сообщении
27    ldi r18,0x17
28 send: ;отправка сообщения
29    lpm r16, Z+
30    sts UDR0,r16
31    subi r18,0x01
32 repeat: ;проверка окончания сообщения
33    lds r17, UCSR0A
34    bst r17, 5
35    brtc repeat
36    cpi r18,0x00
37    breq   main
38    jmp send
39 .cseg ;размещение сообщения в ПЗУ
40 Msg: .db "Atmega328P:ReadyToWork",'\n'
```

Рассмотрим пример более подробно.

Строки 1-21 выполняют инициализацию интерфейса UART в соответствии с примером 1.

Строки 24-25 определяют адрес сообщения в ПЗУ. Для корректного определения адреса значение необходимо сдвинуть на 1 разряд влево.

В строке 26 определяется длина сообщения.

В строке 29 символ загружается в регистр r16, и адрес увеличивается на 1 (переход к другому символу).

Строка 30 – символ записывается в регистр UDR0 на отправку.

Строка 31 – количество символов уменьшается на 1.

В строках 33-35 происходит анализ бита UDRE0 регистра UCSR0A. Если бит равен 1, то вновь осуществляется проверка бита; когда передача символа будет закончена, выполняется проверка количества оставшихся символов.

Строки 36-38 – если символы закончились, то отправка начнется заново.

Строки 39-40 обеспечивают размещение сообщения в памяти ПЗУ. Все символы будут записаны в формате одного байта в соответствии с таблицей ASCII.

Выполните сборку программы. Загрузите код в контроллер. Проанализируйте работу контроллера. Прием информации на компьютере выполните с помощью монитора последовательного порта среды Arduino IDE. Результат зафиксируйте в виде снимка экрана «Монитор последовательного порта».

Задание 2. Разработайте аналогичную программу на языке С. Однако вместо циклического анализа бита UDRE0 сделайте передачу с использованием прерывания, возникающего при событии завершения передачи. Проанализируйте работу программы.

Задание 3. Прием информации: исправьте код примера таким образом, чтобы плата реагировала на принятый байт (см. таблицу 6.8) следующим образом: если получен символ согласно номеру варианта, светодиод L (PortB5) должен зажечься, в противоположном случае – погаснуть. Обработку информации из регистра UDR необходимо вести по прерыванию, иначе, например при считывании значения, в основной программе в цикле может быть получена некорректная инфомрация.

Таблица – 6.8. Варианты для задания 3

№ варианта	1	2	3	4	5	6	7	8	9	10
Символ	A	T	M	e	g	A	3	2	8	P

Для того чтобы контроллер UART принимал данные, необходимо разрешить данную операцию (см. описание регистра UCSR0B). Выполните сборку программы. Загрузите код в контроллер. Проанализируйте работу контроллера. Передачу данных на контроллер выполните с помощью монитора порта Arduino IDE.

Адрес вектора прерывания по завершению приема информации контроллером UART в таблице векторов имеет адрес 0x0028 (имя для

процедуры в Си – USART_TX_vect). Для разрешения прерывания необходимо установить соответствующий бит в регистре UCSR0B. Выполните сборку программы. Загрузите код в контроллер. Проанализируйте работу контроллера. Передачу данных на контроллер выполните с помощью монитора порта Arduino IDE.

Разработайте аналогичную программу на языке С. Повторите вышеперечисленные перечисленные действия.

Задание 4. Обмен пакетами данных.

Разработайте программу, которая получает 4 числа в формате `!a,b,c,d.` (где `a,b,c` и `d` – целые числа, «!» – начало обмена, «.» – завершение обмена), и в порядке `b,a,d,c` передает обратно. Для передачи и приема сообщений используйте соответствующие прерывания. Для разработки программы используйте язык С.

Примечание: при отправке сообщений учитывайте, что монитор порта Arduino IDE работает с числами, как с символами таблицы ASCII, т.е. при передаче символа X контроллер будет работать с числом 0x58, и аналогично с получением.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Поясните разницу между последовательным и параллельным интерфейсом.
2. Опишите последовательность передачи битов в пакете UART.
3. Какие действия необходимо выполнить для инициализации обмена данными по UART в AVR.
4. Почему выполнять обмен данными необходимо с использованием системы прерываний?
5. Какие события контроллера UART могут стать источником прерываний?
6. Как частота обмена данными связана с частотой тактирования контроллера?
7. Из какого минимального и максимального количества бит (включая служебные) состоит передаваемый пакет UART?
8. Какое назначение имеет вывод PortD4 для интерфейса UART?
9. Как понять, что в регистре UDR0 есть данные?
10. Зачем нужен бит четности?