ГЛАВА 3

Аппаратная реализация последовательного интерфейса в компьютерных системах

Все процедуры обмена данными по последовательному интерфейсу осуществляются посредством специальных микросхем, которые получили название универсальных асинхронных приемопередатчиков (от англ. UART — Universal Asynchronous Receiver/Transmitter). Все устройства, которые выполняют последовательный обмен данными, имеют тот или иной тип микросхемы UART. Тем не менее, аппаратная часть большинства таких приемопередатчиков базируется на топологии кристалла 8250 и его более продвинутой версии 8250A. Устройства серии 8250 включают такие популярные микросхемы асинхронных приемопередатчиков, как 16450, 16550, 16650 и т. д. Практически любую из этих микросхем можно обнаружить в персональных компьютерах и других коммуникационных устройствах.

В этой главе мы рассмотрим особенности функционирования аппаратной части устройства UART и программирования обмена данными посредством асинхронного приемопередатчика на нижнем уровне.

3.1. Аппаратная архитектура UART

Асинхронный приемопередатчик является базовым звеном последовательного интерфейса, представляя собой довольно сложный аппаратно-программный модуль обмена данными. В данном разделе мы сосредоточим внимание на функционировании аппаратной части этого устройства. Основные характеристики различных микросхем UART приводятся в табл. 3.1.

Таблица 3.1. *Сравнительные характеристики различных устройств UART*

Микросхема UART	Описание
8250 (8250-B)	Устанавливался на первых моделях персональных компьютеров
16450/(8250-A)	Данное устройство полностью совместимо с UART 8250. Здесь устранены ошибки в регистре разрешения прерываний и включен ряд дополнительных возможностей
16550	В данном устройстве можно использовать внутреннюю буферизацию передаваемых и принимаемых данных посредством буфера FIFO (First In First Out), хотя эта опция работает с ошибками, из-за чего теряются отдельные символы. Обладает более высоким быстродействием по сравнению с микросхемой 16450 и позволяет работать в режиме прямого доступа к памяти (DMA — Direct Memory Access)
16550A (16550AN)	Модифицированная версия UART 16550 (устранены ошибки реализации FIFO). Предпочтительно использовать на повышенных скоростях обмена данными

Микросхемы UART очень широко используются в разработках одноплатных систем автоматизации на базе микропроцессоров и микроконтроллеров. Например, модули последовательного обмена данными всех без исключения современных микроконтроллеров реализованы на принципах работы классических UART, совместимых с 8250. В этой главе мы подробно рассмотрим принципы функционирования и методы программирования асинхронных приемопередатчиков 8250, что даст возможность достаточно глубоко изучить основные принципы реализации последовательного обмена данными и научиться создавать собственные системы передачи данных.

Устройства UART подключаются к шине сигналов/данных микропроцессорной системы. Для классической микросхемы приемопередатчика 8250A схема подключения может выглядеть так, как показано на рис. 3.1.

Приемопередатчик UART 8250A взаимодействует с системой, в которую он подключен посредством шины управления. В качестве такой шины управления может служить шина ISA/EISA/PCI компьютерных систем или специализированная шина систем на базе микроконтроллеров.

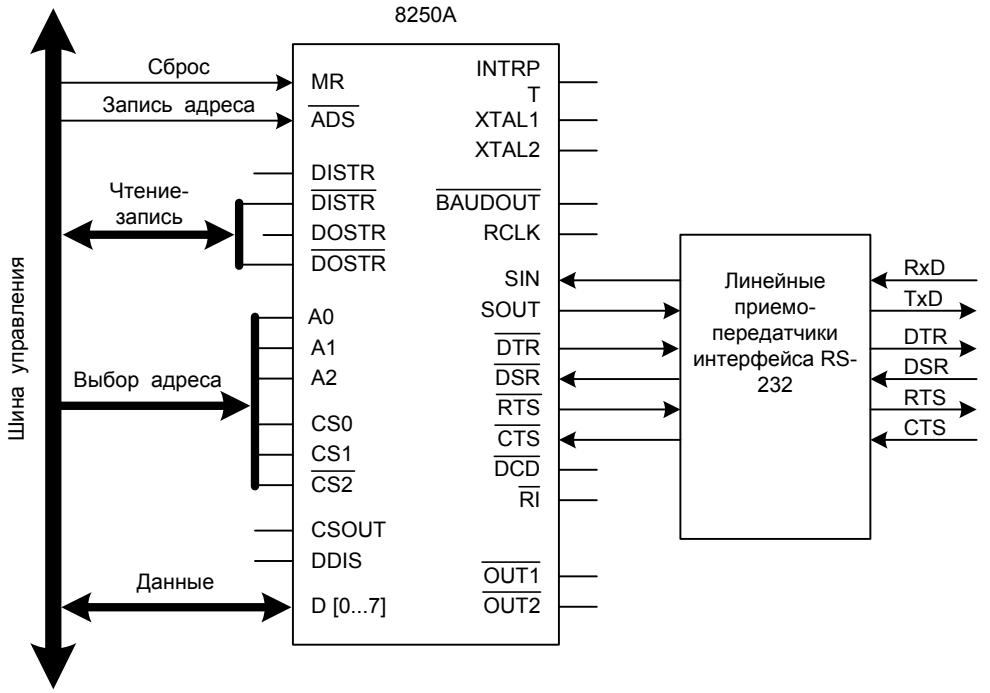

Рис. 3.1. Блок-схема подключения устройства UART 8250A

В устройстве UART можно выделить три относительно независимых функциональных модуля:

- ❒ модуль шинного интерфейса;
- ❒ модуль стандартного приемопередатчика;
- ❒ модуль управления модемом.

Модуль стандартного приемопередатчика обеспечивает передачу данных по линиям TxD и RxD с программным и аппаратным управлением потоком данных. Модуль управления модемом позволяет синхронизировать передачу-прием данных аппаратными средствами (аппаратное управление потоком данных). Модуль управления модемом используется при подключении модемов и другого телекоммуникационного оборудования, где требуется повышенная надежность передачи данных и высокое быстродействие.

Устройство UART имеет целый ряд внутренних программно доступных регистров, посредством которых микропроцессорная система может управлять обменом данными через асинхронный приемопередатчик, а также настраи-

вать параметры обмена данными. Доступ к регистрам UART осуществляется по стандартной схеме обращения к устройствам шины. Например, цикл записи данных в регистр будет выполняться в соответствии с временной диаграммой, показанной на рис. 3.2.

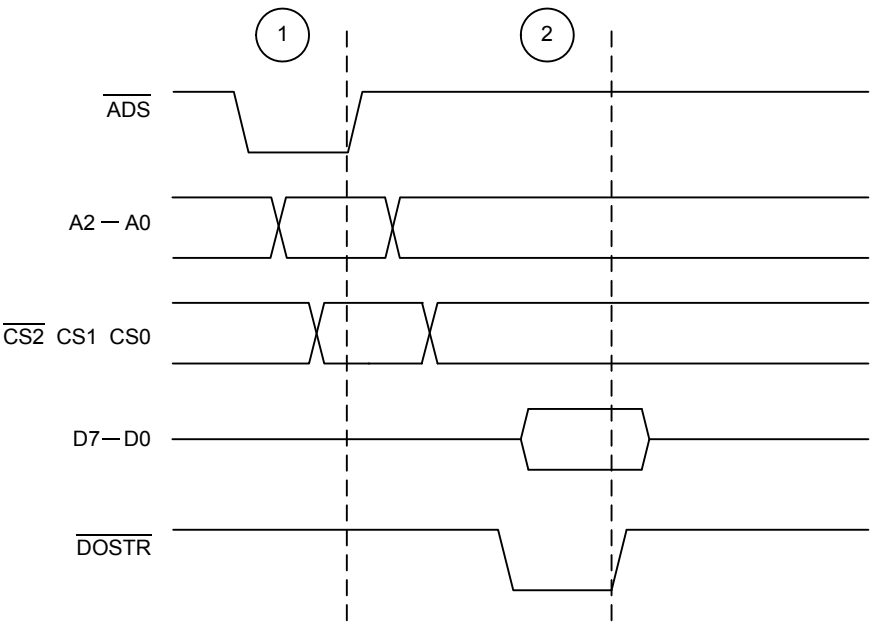

Рис. 3.2. Временная диаграмма записи данных в регистр UART, совместимый с 8250A

Это один из вариантов аппаратной реализации цикла записи байта данных в регистр UART. Здесь для записи используется инверсный строб записи DOSTR, при этом сигнал на линии прямого строба записи должен иметь низкий уровень (2). Можно разработать схему записи данных с использованием прямого строба DOSTR с активным высоким уровнем сигнала, тогда сигнал на инверсной линии должен иметь высокий уровень.

Сигнал ADS с активным низким уровнем служит для фиксации адреса регистра UART, при этом на линиях A0—A2 должен быть установлен адрес регистра (1). Можно обойтись и без строба ADS, но следует гарантировать, чтобы адрес регистра оставался неизменным в течение цикла записи. Линии nCS2, CS1—CS0 служат для выбора кристалла микросхемы и аппаратно могут быть подключены либо к соответствующим уровням напряжений, либо посредст-

вом цифровой логики запрещать/разрешать подачу сигналов чтения-записи. Аппаратная реализация шинного интерфейса в каждом конкретном случае зависит от разработчика.

По такой же схеме выполняется и операция чтения регистра UART, только здесь будут задействованы сигналы DISTR (прямой или инверсный). Кроме рассмотренных сигнальных линий микросхемы UART, 8250A и совместимые с ней имеют и другие выводы, которые должны подключаться в соответствии с технической документацией на каждое конкретное устройство. Аппаратно-программную реализацию шинного интерфейса можно выполнить на микропроцессоре или микроконтроллере, причем сделать это не так и сложно. Все сигналы микросхемы UART работают с уровнями TTL-логики, поэтому проблем по части согласования уровней сигналов на шине не возникает.

Выходные сигналы устройства UART через буферные микросхемы выводятся на внешние выводы разъема последовательного порта. В качестве линейных приемопередатчиков или, по-другому, буферов интерфейса RS-232 в большинстве случаев используется ставшая уже классической микросхема MAX232, хотя в настоящее время она активно заменяется другими устройствами, не требующими дополнительных навесных компонентов.

Все настройки устройства UART, а также прием и передача данных осуществляются посредством регистров, большая часть которых программно доступна пользовательским приложениям, а несколько регистров являются внутренними служебными регистрами и служат для выполнения специализированных операций (например, регистры сдвига приемопередатчика для передачи бита данных в линию TxD или приема бита данных по линии RxD).

Рассмотрим назначение регистров UART. Программисту доступны следующие регистры асинхронного приемопередатчика:

- регистр приема-передачи;
- регистр разрешения прерываний;
- регистр идентификации прерывания;
- регистр управления линией;
- регистр управления модемом;
- регистр состояния линии;
- регистр состояния модема;
- временный регистр.

Каждый из регистров имеет адрес, который отсчитывается от базового адреса устройства UART. В качестве базового адреса принимается адрес регистра приема-передачи. Например, для последовательного порта COM1 персонального компьютера базовый адрес в подавляющем большинстве случаев равен шестнадцатеричному значению 0x3F8, которое является адресом регистра приема-передачи устройства UART. Базовый адрес последовательного порта COM2 обычно равен 0x2F8. Если обозначить адрес базового регистра как PORT1 (такое обозначение мы будем использовать при анализе примеров программирования далее *в этой главе*), то, например, адрес регистра управления линией будет равен PORT1+3, адрес регистра состояния линии будет равен PORT1+5 и т. д.

Регистры UART имеют и специальные мнемонические обозначения. Наиболее часто мы будем использовать регистр управления линией (Line Control Register, LCR), регистр состояния линии (Line Status Register, LSR), регистр управления модемом (Modem Control Register, MCR) и регистр состояния модема (Modem Status Register, MSR). Регистры UART позволяют задавать различные режимы работы и параметры обмена данными. Для изучения практических аспектов программирования устройства UART нам понадобятся только некоторые основные моменты, касающиеся функционирования порта. Сразу замечу, что мы не будем рассматривать расширенные возможности UART, такие как работа с прерываниями и буфером FIFO, поскольку программирование таких функций, особенно в защищенных операционных системах Windows 2000/XP/Vista, требует специальных знаний в области разработки драйверов устройств и глубоких знаний самой операционной системы. Тем не менее, мы проанализируем основы программирования асинхронного приемопередатчика и научимся настраивать параметры обмена данными.

В простейшем варианте для обмена данными по последовательному порту посредством UART следует выполнить такие шаги:

- задать скорость обмена данными устройства UART;
- задать формат данных;
- выполнить передачу или прием данных.

Некоторые из этих шагов можно пропустить, например, не задавать скорость обмена данными, а использовать значение по умолчанию, принятое в системе (обычно скорость обмена устанавливается по умолчанию равной 9600 бод). Можно использовать и стандартное значение формата данных системы (8 бит, 1 стоповый бит, без контроля четности).

Скорость обмена данными задается специальным образом на основе деления тактовой частоты кристалла UART. Стандартная тактовая частота для работы микросхемы UART обычно равна 1,8432 МГц. Внутренняя частота синхронизации будет в 16 раз меньше и равна 1,8432 / 16 = 115,2 кГц. Для получения требуемой скорости обмена в оба байта делителя частоты нужно записать значение, равное частоте синхронизации в герцах деленной на скорость обмена в бодах. Например, для стандартной скорости обмена, равной 9600 бод, в регистры делителя частоты нужно записать значение 115200 / 9600 = 12 (или 0x0C в шестнадцатеричной нотации). Таким образом, в младший байт делителя частоты будет записано значение 0x0C, а в старший байт — 0x0. Если значение делителя частоты установить равным 1, то получим максимальную скорость обмена 115200 бод.

В табл. 3.2 приводятся некоторые значения делителя скорости для ряда стандартных скоростей обмена.

Таблица 3.2. *Значения делителя частоты для разных скоростей обмена*

Скорость (в бодах)	Десятичное значение	Шестнадцатеричное значение
50	2304	0900
300	384	0180
600	192	00C0
1200	96	0060
2400	48	0030
4800	24	0018
9600	12	000C
19200	6	0006
38400	3	0003
57600	2	0002
115200	1	0001

Первое, что нужно сделать для настройки скорости обмена — установить бит 7 регистра управления линией (LCR) в 1. Этот бит имеет специальное

обозначение DLAB (Divisor Latch Access Bit) и позволяет получить доступ к 16-битовому регистру делителя скорости. Он состоит из младшей 8-битовой части (DLL, Divisor Latch Low), которая доступна по базовому адресу UART, и старшей части (DLH, Divisor Latch High), доступной по базовому адресу + 1.

После установки бита DLAB нужно записать требуемое значение в оба 8-битовых регистра делителя. После установки скорости обмена следует очистить бит DLAB, иначе он будет блокировать доступ к регистру приема-передачи, что сделает обмен данными невозможным.

Следующий этап — установка формата данных. Для настройки этого параметра используются биты 0—3 регистра управления линией (LCR, базовый адрес + 3). В табл. 3.3 показаны значения этих битов для 7- и 8-битовой посылки данных с одним стоповым битом и без контроля четности.

Таблица 3.3. *Настройка формата данных в регистре управления линией*

Формат данных	**Бит 3 (контроль четности)**	**Бит 2 (количество стоповых бит)**	**Бит 1**	**Бит 0**
7 бит, 1 стоповый, без контроля четности	0	0	1	0
8 бит, 1 стоповый, без контроля четности	0	0	1	1

Таким образом, для установки 8-битовой посылки, с одним стоповым битом и без контроля четности в регистр LCR нужно записать значение 0x3.

Прием-передача данных осуществляется посредством базового регистра приемопередатчика. При этом для передачи байта нужно записать его значение в регистр приема-передачи, а при приеме данных — прочитать значение, находящееся в этом регистре. В процессе обмена данными нужно точно знать, когда очередной байт данных отправлен из регистра передачи в линию (для посылки следующего байта) или когда очередной байт находится в регистре приема. О полном завершении процесса передачи байта данных будет свидетельствовать установленный бит 6 регистра состояния линии (LSR). О наличии байта данных в регистре приема будет свидетельствовать установленный бит 0 регистра LSR. Оба бита сбрасываются устройством автомати-

чески при записи очередного байта данных для передачи или при чтении полученного байта данных из регистра приема.

Все описанные ранее операции мы реализуем на практических примерах, которые приведены далее в *этой главе*.

3.2. Диагностика и настройка интерфейса RS-232

В процессе разработки приложений, для которых требуется выполнять обмен данными по интерфейсу RS-232, часто возникает необходимость настройки и проверки функционирования последовательного порта того или иного устройства на уровне аппаратных средств, т. е. асинхронного приемопередатчика (UART). Это касается как персональных компьютеров, так и специализированных устройств. Для персональных компьютеров, работающих под управлением операционных систем Windows, такую настройку и диагностику последовательного порта выполнить несложно, используя арсенал современных средств разработки (Ассемблер, Delphi, Microsoft Visual Studio и т. д.). То же самое касается и популярных операционных систем Linux, которые функционируют на многих ПК и в большинстве телекоммуникационных устройств, хотя там имеются и некоторые особенности, которые мы проанализируем позже.

В простейшем варианте тестирования и настройки COM-порта нужно соединить выводы сигнальных линий TxD и RxD на разъеме последовательного порта ПК (рис. 3.3).

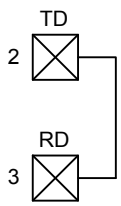

Рис. 3.3. Шлейф обратной связи для тестирования COM-порта

В этом случае можно программным способом отправлять данные по линии TxD и тут же принимать их по линии RxD одного и того же порта. Этот метод очень удобен, поскольку не требуется никакого дополнительного обору-

дования, подключенного к COM-порту. Естественно, при таком методе проверки имеются и некоторые ограничения. Например, сложно проверить работу интерфейса в реальном времени, особенно при использовании сигналов квитирования (подтверждения), как это часто требуется для реальных приложений. Тем не менее, такое тестирование позволяет довольно быстро определить работоспособность интерфейса и его функционирование при задании различных параметров обмена данными.

Если под рукой имеется демо-плата на базе микроконтроллера или микропроцессора с интегрированным на ней последовательным портом, то ее можно использовать для тестирования или настройки интерфейса последовательного обмена данными.

При разработке программной части проверки и настройки микросхемы UART интерфейса RS-232 следует учитывать особенности функционирования операционных систем, под управлением которых работает последовательный интерфейс. В следующих разделах мы проанализируем особенности настройки и тестирования асинхронного приемопередатчика последовательного интерфейса в различных операционных системах. Начнем с настройки UART в операционных системах Windows 98/Me. Эти операционные системы благодаря возможности прямого доступа к аппаратным средствам широко используются для настройки и управления периферийным оборудованием с интерфейсом RS-232, особенно в лабораторных и промышленных системах сбора и обработки данных. Это не в последнюю очередь связано и с тем, что фирма Microsoft выпустила программное обеспечение (Virtual PC 2004/2007) для создания различных виртуальных операционных систем, включая Windows 98/Me, работающих в установленных на ПК системах Windows XP/2003/Vista. Самое существенное это то, что при работе в таких виртуальных системах пользователь получает прямой доступ к аппаратным ресурсам параллельного и последовательного портов без сложной и трудоемкой процедуры написания драйверов ядра, а это открывает широкие возможности по адаптации и настройке оборудования относительно простыми средствами.

3.2.1. Настройка и тестирование UART в операционных системах Windows 98/Me

Для настройки/тестирования устройства UART в операционных системах Windows 98/Me мы будем использовать две наиболее популярные программные среды разработки — Microsoft Visual Studio и Delphi. В данном случае

мы будем использовать Delphi 7 и Visual C++ 6 пакета Microsoft Visual Studio 6, хотя программы можно компилировать и в других версиях этих программных продуктов.

Для настройки/тестирования асинхронного приемопередатчика COM-порта ПК можно соединить выводы TxD и RxD последовательного порта или же подключить к последовательному порту устройство, которое могло бы принимать данные из порта или пересылать их в порт. Программное обеспечение пользователя может либо читать данные из последовательного порта, отображая их на экране консоли, либо отправлять данные в порт присоединенному устройству.

Конечно, использование замкнутого шлейфа TxD—RxD ("заглушки") позволяет при минимальных усилиях проверить и настроить большинство параметров обмена данными, а также проверить работу самого порта. Тем не менее, следует учитывать, что в реальных системах должны выдерживаться определенные временные зависимости между сигналами интерфейса приемного и передающего устройства, поэтому окончательную настройку процедуры обмена данными желательно все же выполнять с реальным оборудованием, подключенным к последовательному порту.

Тем не менее, использование замкнутого шлейфа дает программисту возможность без установки оборудования достаточно хорошо изучить принципы обмена данными. Это все, что касается аппаратных подключений. Теперь посмотрим, что нужно для программной части. Во-первых, на персональном компьютере нужно проинсталлировать одну из операционных систем Windows 98/Me. Поскольку в настоящее время многие пользователи уже имеют установленную версию одной из операционных систем Windows 2000/XP/Vista, то можно пойти и другим путем. В этом случае лучше всего использовать бесплатно предоставляемую Microsoft программную среду Virtual PC 2007, с помощью которой можно без особого труда установить "виртуальную" полнофункциональную Windows 98/Me. Естественно, нужно иметь дистрибутив Windows 98/Me. Установка "виртуальной машины" довольно проста и достаточно хорошо описана в документации Microsoft.

Во-вторых, в операционной системе Windows 98/Me нужно установить предпочтительную среду разработки (Delphi или Visual Studio). Наконец, желательно иметь установленную программу терминального доступа по последовательному порту — это не проблема, поскольку имеется великое множество таких программ. В принципе, можно обойтись и без такой программы, но ее использование существенно упрощает тестирование/настройку.

Перейдем к разработке программного обеспечения для тестирования/настройки последовательного порта, вернее, его аппаратной части — асинхронного приемопередатчика (UART). Начнем с примеров обмена данными, разработанных в среде Delphi 7. Первое приложение будет отправлять строку байтов терминальной программе, запущенной на ПК при соединенных сигнальных линиях TxD и RxD. Эту схему мы будем часто использовать при настройке/тестировании последовательного порта (рис. 3.4).

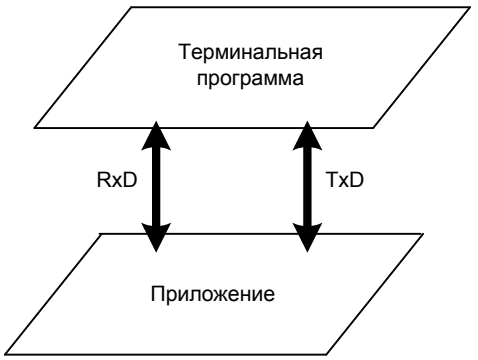

Рис. 3.4. Схема взаимодействия приложения и терминальной программы

Вместо терминальной программы для настройки порта можно использовать другое приложение. В этом случае одна пользовательская программа отправляет данные, а другая — их принимает.

Вид окна конструктора приложения показан на рис. 3.5.

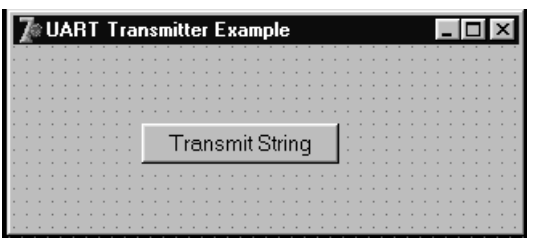

Рис. 3.5. Вид окна конструктора приложения

Приложение работает предельно просто: при нажатии кнопки **Transmit String** терминальной программе отправляется строка байтов. Вся работа про-

граммы выполняется функцией-обработчиком нажатия кнопки. Исходный текст программы приведен в листинге 3.1.

Листинг 3.1. Передача строки байтов посредством UART

```
 unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  s1: PChar;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
const
 PORT1 = $3F8;
 LSR = $3FD;
```

```
begin
 s1:= 'Hello,Dumb Terminal! ';
 asm
  mov  ESI, dword ptr s1
@next_ch:
  mov AL, byte ptr [ESI]
  mov DX, 3F8h

  cmp AL, 0
  je  @quit
  out DX, AL

  mov DX, 3FDh
@wait_trm:
  in  AL, DX
  and AL, 40h
  cmp AL, 40h
  jne @wait_trm
  mov BL, 100
@delay:
  dec BL
  jnz @delay
  inc ESI
  jmp @next_ch
@quit:
 end
end;

end.
```

Программа передает строку байтов (переменная `s1`) посредством прямого программирования регистров устройства UART в процедуре `Button1Click`. Процедура написана на встроенном ассемблере — это позволяет лучше почувствовать работу аппаратуры, хотя при желании можно ее переписать с помощью операторов высокого уровня. Адрес строки помещается в ре-

гистр ESI, посредством которого осуществляется доступ к отдельным ее элементам:

```
mov  ESI, dword ptr s1
```

В каждой итерации байт помещается в регистр AL, где он проверяется на равенство нулю. В Delphi строка типа PChar заканчивается нулем, что и будет служить признаком окончания передачи. Затем, если символ в регистре AL не равен 0, то командой out он выводится в базовый порт приемопередатчика, который содержится в регистре DX (в данном случае выбран стандартный адрес, равный 0x3F8). Описанные действия выполняются следующей группой команд:

```
mov AL, byte ptr [ESI]
mov DX, 3F8h
. . .
cmp AL, 0
je  @quit
out DX, AL
```

Команда out записывает байт данных из регистра AL в регистр приемопередатчика UART, который является базовым регистром COM-порта. Байт данных из регистра передатчика записывается в регистр сдвига (это внутренняя операция UART), откуда синхронно с тактовыми импульсами синхронизации биты выдвигаются в линию TxD. Процесс передачи байта считается полностью законченным, когда регистр приемопередатчика и регистр сдвига будут пустыми, о чем свидетельствует установленный бит 6 регистра состояния линии LSR (PORT1 + 5). Установленный бит 5 этого регистра свидетельствует о том, что регистр передатчика пуст, но регистр сдвига продолжает передачу битов на линию TxD.

Следующий байт можно передавать при любом установленном бите 5 или 6. В нашей программе проверяется состояние бита 6 регистра LSR:

```
  mov DX, 3FDh
@wait_trm:
  in  AL, DX
  and AL, 40h
  cmp AL, 40h
  jne @wait_trm
```

После передачи байта (регистр передачи и регистр сдвига пусты) в программе выполняется некоторая задержка с помощью команд

```
@delay:
  dec BL
  jnz @delay
```

В принципе, в реальном приложении такая задержка может и не понадобиться, но в данном случае программа-терминал и программа-передатчик работают с одним и тем же портом, поэтому между передачей символов делается определенная задержка, что позволяет избежать переполнения (особенно если FIFO-буфер отключен). В любом случае можно поэкспериментировать с настройкой режима передачи с задержкой и без нее.

По окончании передачи байта указатель смещается к следующему символу, и цикл повторяется до тех пор, пока не будет обнаружен признак конца строки (нулевой байт):

```
inc ESI
jmp @next_ch
```

Замечу, что в этой программе не выполнялась настройка параметров передачи. Эти параметры берутся по умолчанию из настроек системы или настроек терминальной программы.

Следующий пример показывает, как можно принимать байты данных по последовательному порту посредством прямого доступа к регистрам UART. В этом примере символы, вводимые в окне терминальной программы, будут отображаться в окне приложения. Как и в предыдущем примере, для тестирования следует соединить линии TxD и RxD.

Вид конструктора окна приложения показан на рис. 3.6.

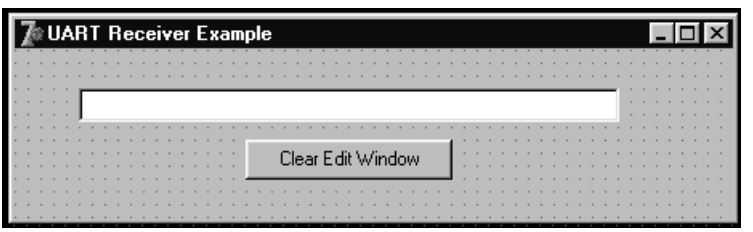

Рис. 3.6. Вид окна конструктора приложения

Программа чтения данных с последовательного порта сложнее, чем та, которую мы анализировали в предыдущем примере. Для ожидания приема байта в программе нужно создать отдельный поток, который будет непрерывно анализировать бит 0 регистра состояния линии (LSR) на равенство 1. Как только бит 0 будет установлен в 1, это будет означать, что данные из входного регистра сдвига поступили в регистр приемника (PORT1) и могут быть прочитаны. Прочитанный байт отображается в окне редактирования, а курсор смещается вправо в ожидании нового символа. Очистка буфера окна осуществляется при нажатии кнопки **Clear Edit Window** (см. рис. 3.6).

Бит 0 регистра LSR при чтении регистра приемника (PORT1) автоматически сбрасывается, поэтому в программе нет необходимости делать это вручную. Исходный текст программы приведен далее в листинге 3.2.

Листинг 3.2. Прием байтов из последовательного порта посредством UART

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Button1: TButton;
    procedure FormActivate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure UART_init;

  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
 type
  Thread1 = class(TThread)
  protected
    procedure Execute; override;
    procedure UpdateEdit;
 end;

const
 PORT1 = $3F8;
 BRH   = PORT1 + 1;
 FIFO  = PORT1 + 2;
 LCR   = PORT1 + 3;
 MCR   = PORT1 + 4;
 LSR   = PORT1 + 5;

var
  myThread: Thread1;
  Form1: TForm1;
  c1: Byte;
implementation

{$R *.dfm}
procedure TForm1.UART_init;
begin
 asm
  mov  DX, BRH
  xor  AL, AL
  out  DX, AL

  mov  DX, LCR
  mov  AL, 80h
  out  DX, AL

  mov  DX, PORT1
  mov  AL, 0Ch
  out  DX, AL
```

```
  mov  DX, BRH
  xor  AL, AL
  out  DX, AL

  mov  DX, LCR
  mov  AL, 3h
  out  DX, AL

  mov  DX, FIFO
  xor  AL, AL
  out  DX, AL

  mov  DX, MCR
  out  DX, AL
 end;
end;

procedure Thread1.Execute;
begin
 while True do
  begin
   asm
     mov DX, LSR
    @again:
     in  AL, DX
     and AL, 1h
     cmp AL, 1
     jne @again
     mov DX, PORT1
     in  AL, DX
     mov c1, AL
   end;
     Synchronize(UpdateEdit);
  end;
end;
procedure Thread1.UpdateEdit;
```