Елена Бадло, Сергей Бадло
г. Запорожье
E-mail: **raxp@radioliga.com**

Продолжая цикл статей по начальной практике программирования, сегодня мы с вами создадим виртуальный монитор LPT порта с визуальной индикацией состояния линий ввода-вывода и управляющих регистров.

Виртуальные приборы. Виртуальный монитор LPT порта

Предугадываем резонный вопрос читателя: “Насколько вообще актуально сейчас писать о LPT интерфейсе, когда в большинстве современных машин он не используется вовсе, а то и физически отсутствует?”. И будет прав. Безусловно, LPT порт отживает свое, не спорим. Но остается довольно большой парк “старых” системников, используемых в промышленности для задач автоматики, например для ЧПУ, да и самими радиолюбителями при программировании и управлении через этот интерфейс. При этом часто возникает необходимость в простом мониторе его состояния, работающим как в средах Windows NT, так и Windows 9x.

Краткий экскурс...

И если в старой доброй среде Win 9x у пользователя не возникает проблем “как работать с портами” (доступ открыт), то в NT-подобных системах эта возможность отсутствует и приходиться прибегать к услугам отдельного драйвера или посреднических API функций “оси”, както – *CreateFile*, *ReadFile* и т.д. Но как только мы их используем, то сразу теряем возможность работы в старых системах, так как эти функции в них не работают.

Именно для этих случаев и существует бесплатный драйвер – GiveIO.sys. Главным достоинством этого драйвера является то, что обращения к портам могут быть сделаны и с помощью функций, работающих только в Windows 9x. Давным-давно этот драйвер написал Dale Roberts, за что ему благодарны многие поколения как программистов, так и радиолюбителей. Рассмотрим вкратце, как работает этот драйвер…

Предпосылки реализации ПО. Существующие решения

Прежде всего, в начале работы нашего приложения для обмена информацией с портами необходимо обратиться к драйверу – “GiveIO.sys”. При этом он установит такие значения битового массива карты разрешения (I/O Permission Map) для нашей программы (процесса в памяти), что для него будет разрешен прямой доступ к любым портам.

Для чего нужен этот I/O Permission Map? I/O Permission Map – это составная часть системы защиты ввода-вывода в средах Windows NT. Он представляет собой битовый массив, каждый бит которого соответствует порту ввода-вывода. К примеру, если бит равен 1, то доступ к порту закрыт, если бит равен 0 – открыт. Для любого пользовательского приложения все биты установлены в 1 и, соответственно, мы не можем напрямую влезть в порт. В то же время драйвер GiveIO.sys имеет доступ в этот

Рис. 1. “Мониторим” порты

массив и везде устанавливает 0, при этом мы получаем возможность обратиться к любому порту из приложения.

Выводы LPT порта можно разделить на четыре группы (см. **рис. 2**):

- “общие” (земляные) – контакты с 18 по 25;
- двунаправленный программируемый регистр данных DATA (базовый адрес = $378) – контакты с 2 по 9;
- однонаправленный (управление только извне) регистр STATUS (базовый адрес = $379) – контакты 10, 11, 12, 13, 15;
- однонаправленный (управление только программно) регистр CONTROL (базовый адрес = $37A) – контакты 1, 14, 16, 17.

Исходя из вышеизложенного, уже можем сформировать основные требования к нашему монитору:

- автосоздание драйвера прямого доступа к портам (как составная часть модуля);
- опрос и индикация состояний всех пинов LPT;
- возможность изменить режим каждого из регистров;
- возможность удержания выбранного состояния регистра управления.

Практика. Разработка ПО и средства отладки

Итак, приступим к основной задаче. Для работы нам понадобиться следующее:

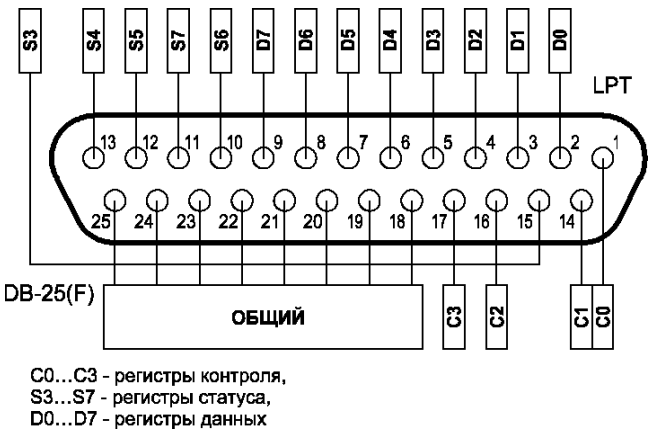

Рис. 2. Распиновка LPT

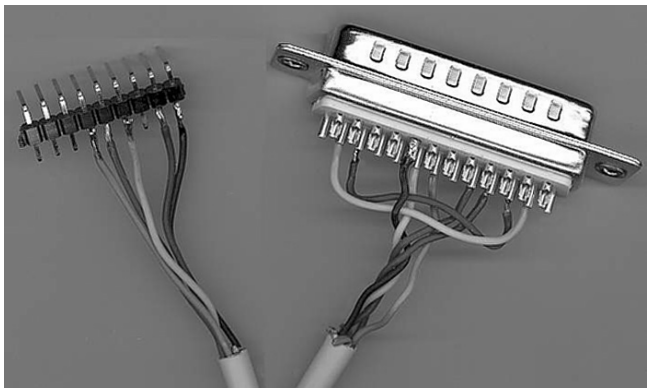

Рис. 3. Удлинитель-переходник

- среда Borland Delphi 5-7 (компиляция и отладка проекта) или бесплатная альтернатива от Borland – Delphi 10.Lite [1]
- модуль прямого доступа к портам через драйвер GIVEIO.sys разработки автора [2]
- мультиметр или набор светодиодов, батарейка 4,5 В, разъем-вилка DB-25, резистор на 1 кОм для проверки состояния пинов на LPT интерфейсе

Для удобства проверки и тестирования лучше смонтировать простейший удлинитель-переходник с вилкой DB-25 на конце, например такой, как на **рис. 3**.

Автосоздание драйвера

Настало время взяться за программирование. Ввиду ограниченности места в журнале, рассмотрим только основные моменты реализации. Как мы уже упоминали, необходимо вначале зарегистрировать и установить драйвер в системе. Для этого необходимо сделать следующие шаги:

1. Открываем текст драйвера GIVEIO.sys в блокноте и копируем его содержимое прямо в наш модуль в виде массива. Должно получиться нечто вроде такого (см. **врезку**).

врезка

```
A : Array[1..5248] of Byte=(77,90,144, 0,3,0,0,0,4,0,0,0,255,255,0,0,184, ... )
```

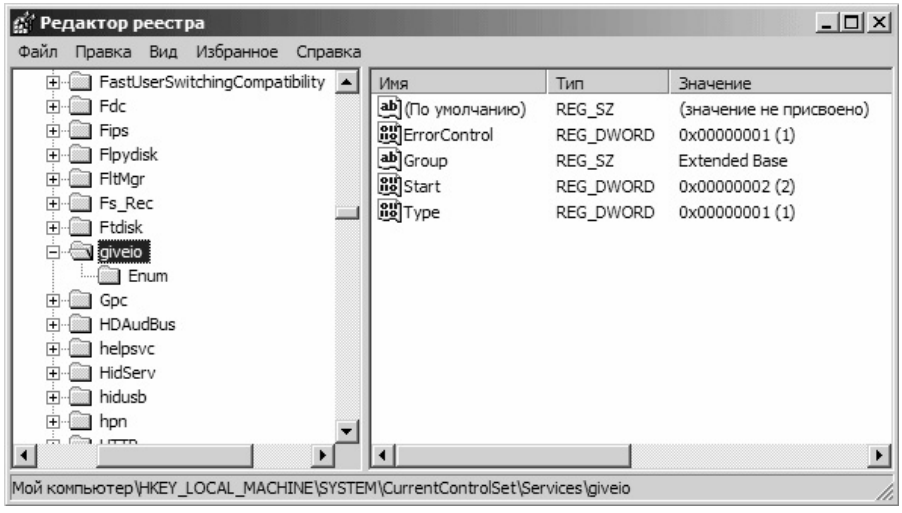

Рис. 4. Регистрация драйвера в системе. Ветка реестра

2. При создании проекта в событие *OnCreate* формы вызываем процедуру сохранения данного массива в виде файла API функциями *AssignFile()*, *ReWrite()*, *BlockWrite()*. При этом в каталоге с нашей программой создастся и сам драйвер. Вот такой вот "хитрый" способ.

3. И самое главное, не забыть зарегистрировать драйвер в системе. Для чего подключим в uses проекта – модуль *WinSvc*. В результате мы сможем воспользоваться известными командами для создания, регистрации и старта сервиса: *OpenSCManager()*, *CreateService()*, *StartService()* (*см. исходники*). После чего, открыв соответствующую ветку в реестре, можем убедиться в правильной установке и регистрации драйвера (см. **рис. 4**).

Пользовательский интерфейс

Рассмотрим создание формы. Прежде всего, в IDE с закладки *Addition* переместите на форму 17 компонентов *Shape*. Установите тип контура – круг. Эти компоненты и будут нашими индикаторами, для чего достаточно будет изменять их цвет. Зеленый – при установке бита в "1" и белый – при "0". Для управления регистрами будем обрабатывать нажатия левой кнопкой мыши на каждом из индикаторов, вызывая опрос или запись в порт соответственно.

Собственно, известные из Win 9x функции прямого обращения к портам (см. **листинг 1**).

Установку состояний регистров обмена, контроля и управления будем осуществлять при "клике" по визуальной картинке каждого пина LPT (см. **листинг 2**).

Опрос состояний регистров обмена, контроля и управления будем осуществлять в потоке таймера (см. **листинг 3**).

прямая адресация к портам **листинг 1**

```
…
 function getbit(data,num:byte): boolean;
 begin
  result:= data and num = num
 end;

 //регистр обмена-
 function gdLPT: byte;
 begin
  result:= FInByte(BA)
 end;
 procedure sdLPT(data: byte);
 begin
  FOutByte(BA,data)
 end;

 //регистр контроля-
 function gkLPT: byte;
 begin
  result:= FInByte(BR)
 end;
 procedure skLPT(data: byte);
 begin
  FOutByte(BR,data)
 end;

 //регистр управления-
 function guLPT: byte;
 begin
  result:= FInByte(BU)
 end;
 procedure suLPT(data: byte);
 begin
  FOutByte(BU,data)
 end;
…
```

листинг 2

установка регистров

```
…
procedure setd; // регистры данных-
var i: integer;
    temp: byte;
begin
 temp:=0; //обмен-
 for i:=0 to 7 do
  if b[i].Brush.Color = cllime then temp:= (1 shl i) or temp
   else temp:= (0 shl i) or temp;
 sdLPT(temp)
end;

procedure setk; // регистры контроля-
var i: integer;
    temp: byte;
begin
 temp:=0; //контроль-
 for i:=8 to 11 do
  if b[i].Brush.Color = cllime then temp:= (1 shl (i-8)) or temp
   else temp:= (0 shl (i-8)) or temp;
 skLPT(temp)
end;

procedure setu; // регистры управления-
var i: integer;
    temp: byte;
begin
 temp:=0;
 for i:=12 to 16 do
  if b[i].Brush.Color = cllime then temp:= (1 shl (i-12+3)) or temp
   else temp:= (0 shl (i-12+3)) or temp;
 suLPT(temp)
end;

procedure tlptf.to_ser(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
 //триггер цвета-
 if b[(sender as tshape).tag].Brush.Color = clwhite then b[(sender as tshape).tag].Brush.Color:= cllime
  else b[(sender as tshape).tag].Brush.Color:= clwhite;
 // при клике на ячейке передаем управление состоянием-
 setd; setk; setu
end;
…
```

листинг 3

опрос по таймеру состояния линий LPT

```
…
 //удержание регистров-
 if lptf.rgd.Checked then setd;
 if lptf.rgk.Checked then setk;
 if lptf.rgu.Checked then setu;

 //регистр обмена-
 bit:= gdLPT;
 for i:=1 to 8 do
  if (bit and (1 shl (i-1))) > 0 then b[i-1].Brush.Color:= cllime
   else b[i-1].Brush.Color:= clwhite;

 //регистр контроля-
 bit:= gkLPT;
 for i:=1 to 4 do
  if (bit and (1 shl (i-1))) > 0 then b[i-1+8].Brush.Color:= cllime
   else b[i-1+8].Brush.Color:= clwhite;

 //регистр управления-
 bit:= guLPT;
 for i:=4 to 8 do
  if (bit and (1 shl (i-1))) > 0 then b[i-1+9].Brush.Color:= cllime
   else b[i-1+9].Brush.Color:= clwhite;
…
```

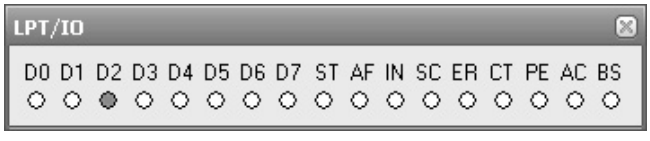

Рис. 5. Виртуальный монитор.
Индикация состояний регистров

После компиляции тестового проекта монитора, запустив приложение, проведем диагностику состояний линий, для чего можно подать на один из пинов шины данных LPT через ограничительный резистор потенциал порядка 4,5 В. При этом должен “засветиться” соответствующий индикатор (см. **рис. 5**).

Для проверки программного управления достаточно подключить щупы мультиметра в режиме измерения напряжения на соответствующий пин шины и общий (земляной) провод и “кликнуть” кнопкой мыши по индикатору этого пина.

Заключение

Удачи в управлении и диагностике!

Полные исходные тексты, ресурсы проекта и модуль прямого доступа к портам (файл *lpt_res.zip*) вы можете загрузить с сайта нашего журнала

http://www.radioliga.com (раздел “Программы”)
а также с сайта автора:
http://raxp.radioliga.com

Если тема представляет для вас интерес – пишите, задавайте вопросы на форуме:
http://raxp.radioliga.com/forum

Ресурсы

1. Бесплатный IDE от Borland – Delphi10 - http://r5.letitbit.net/download5/19b766322354_8avxbh13cklanwkx/Delphi10-Lite-v3.0.rar
2 Компонент прямого доступа к портам DELPHI6 - http://raxp.radioliga.com/cnt/s.php?p=raport.zip
3. Исходники тестового монитора диагностики и управления - http://raxp.radioliga.com/cnt/s.php?p=lpt_res.zip