Dynamic C

IP Driver Manual

001106-A

Dynamic C IP Driver Manual

Part Number 019-0089 • 001106-A
Last revised on November 6, 2000 • Printed in U.S.A.

Copyright

Trademarks

Notice to Users

When a system failure may cause serious consequences, protecting life and property against such consequences with a backup system or safety device is essential. The buyer agrees that protection against consequences resulting from system failure is the buyer's responsibility.

This device is not approved for life-support or medical systems.

All Z-World products are 100 percent functionally tested. Additional testing may include visual quality control inspections or mechanical defects analyzer inspections. Specifications are based on characterization of tested sample units rather than testing over temperature and voltage of each unit. Rabbit Semiconductor may qualify components to operate within a range of parameters that is different from the manufacturer's recommended range. This strategy is believed to be more economical and effective. Additional testing or burn-in of an individual unit is available by special arrangement.

Company Address

Z-World, Inc.

2900 Spafford Street
Davis, California 95616-6800
USA

Telephone: (530) 757-3737
Facsimile: (530) 753-5141
Web site: http://www.zworld.com

Table of Contents

1. PPP Driver

1.1 Introduction

The PPP packet driver is a set of libraries in Dynamic C that allow the user to establish a PPP(Point-to-Point Protocol) link over a full-duplex serial line between a Rabbit-based controller and another system that supports PPP. The established link supports the transfer of Internet Protocol (IP) data and is compatible with all of the Transmission Control Protocol/ Internet Protocol (TCP/IP) libraries for the Rabbit.

The Rabbit PPP library supports both authentication of peers and being authenticated by a peer using Password Authentication Protocol (PAP). This is a simple two-way handshake only done upon initial link establishment.

The PPP packet driver was derived from source code originally written by Darby Corporate Solutions (DCS).

1.2 Libraries

The PPP driver is in two library files.

`PPPLINK.LIB` contains

- The interrupt service routine for transmitting and receiving characters over the serial link. It also handles the insertion and detection of escape characters and CRC generation and checking.
- **`PPPflowcontrolOn()`**, **`PPPflowcontrolOff()`**, and **`PPPclose()`**

`DCSPPP.LIB` contains

- Routines for setting up and running the PPP connection.
- Code for the state machine that handles negotiation of the connection with the peer.

A third library, **`EXTERNAL_MODEM.LIB`**, contains functions for controlling an external modem through a full RS232 link.

1.3 Operation Details

The PPP link works through serial port C. This cannot be changed.

Flow control is usually required for baud rates above 9600.

One of the most common uses of the PPP protocol is the transfer of IP packets between a remote host and an Internet Service Provider (ISP) over a modem connection. For directly connecting a serial line to the peer, the two serial data lines may be adequate for low speeds (9600 baud max). Higher speed connections through a direct line or modem will usually require flow control. Hardware flow control is implemented for the Rabbit PPP system. It follows the RS232 convention of using Ready To Send (RTS) and Clear To Send (CTS) lines. If a modem is used, the additional control signals for the RS232 standard should be connected.

The modem control library, **`EXTERNAL_MODEM.LIB`**, defines default connections to the Rabbit as follows:

1.4 Software

compile time, it can be set up as a gateway to other hosts if that is within the peer's capabilities.

The following configuration options are supported by the Rabbit PPP system:

Table 2. Configuration Options

01	**MKU**
02	**ACCM**
03	**Auth (PAP only)**
05	**Magic Number**
07	**PFC**
08	**ACFC**

As mentioned before, one of the difficulties with dial-up PPP is an ISP will try to authenticate the dialer before PPP negotiation. There are no real standards for doing this, so each ISP is potentially different. The best way to develop a correct sequence of **`ModemSend()`** and **`ModemExpect()`** commands is to connect to the ISP using a terminal program on a PC. You can then take note of the necessary sequence to start PPP negotiation.

Here is a hypothetical session as seen by a terminal program. Note, characters typed in and sent to the ISP or the modem are displayed in **`bold`**.

```
AT
OK
ATDT5554545
OK
CONNECT 28800
Welcome to someisp.com
Login?rabbit
Password:Ilikecarrots
Logging in as rabbit
Start PPP $*($}}}}}$}$#$#${@#>>}}FF}}$}
```

From this session we could use **ModemSend()** and **ModemExpect()** to create a dial-up function like this:

```
int myDialUp()
{
   if(ModemOpen(57600) == 0)
   {
      return 0;
   }
   if(ModemInit() == 0)
   {
      return 0;
   }
   ModemSend(“ATDT5554545\r”);
   if (ModemExpect(“OK”, 2000) == 0))
   {
      return 0;       //something is wrong with the modem
   }
   if(ModemExpect(“CONNECT”, 30000) == 0)
   {
      return 0;       //didn’t connect to the ISP
   }
   if(ModemExpect(“Login?”, 5000) == 0)
   {
      return 0;
   }
   ModemSend(“rabbit\r”);
   if(ModemExpect(“word:”, 5000) == 0)
   {
      return 0;
   }
   ModemSend(“Ilikecarrots\r”);
   if(ModemExpect(“PPP”, 5000) == 0)
   {
      return 0;       //probably a failed login
   }
   ModemClose();
   sock_init();
   PPPinit(57600);
   PPPflowcontrolOn();
   return 1; //all done
}
```

As you can see, **ModemExpect()** will pick up any part of the received string. Clever use of this allows the initialization to be fairly generic, but subtle differences between ISP’s will often require customized sequences such as this.

1.4.1 Link Teardown

Tearing down the link must also be done in stages. First, a terminate request must be sent to the peer. This is done with **PPPshutdown()**. **PPPshutdown()** will return once an acknowledgement has been sent by the peer, or after a time out period. This is followed by a call to **`PPPclose`**, which unloads the PPP serial driver. If the connection is via a modem, the modem must then be hung up. First the regular serial driver is reopened with **`Mode-mOpen()`**. **`ModemHangup()`** sends the hang up and reset commands to the modem. Finally, a call to **`ModemClose()`** shuts down the serial driver.

1.5 Functions

This section describes the functions that compose the PPP driver and the functions for modem control.

1.5.1 Using Cofunctions

Establishing a PPP connection over a modem is time-consuming. Depending on the baud rate negotiated by the modem, the whole process can take 30 seconds or more. Much of this time is spent by the controller waiting for a response from the other end. In a practical application where the controller has other tasks to perform, this may be unacceptable. For this, there are cofunction versions of all of the functions that wait for responses from the peer. There are still parts of the initialization process that create delays, but the effect is much smaller.

CofModemExpect

Syntax

```
int CofModemExpect(char *send_string, unsigned long timeout);
```

Description

Listens for a specific string to be sent by the modem. Yields to other tasks while waiting for input.

Parameters

`send_string`

A null-terminated string to listen for.

`timeout`

Maximum wait in milliseconds for a character.

Return value

1 if the expected string was received
0 if a timeout occured before receiving the string

Library

`EXTERNAL_MODEM.LIB`

CofModemHangup

Syntax

```
int CofModemHangup();
```

Description

Sends "ATH" and "ATZ" commands. Yields to other tasks while for responses.

Parameters

None.

Return value

1 - success
0 - modem not responding

Library

```
EXTERNAL_MODEM.LIB
```

CofModemInit

Syntax

```
int CofModemInit();
```

Description

Resets modem with AT, ATZ commands. Yields to other tasks while waiting for responses.

Parameters

None.

Return value

1 - success
0 - modem not responding

Library

```
EXTERNAL_MODEM.LIB
```

CofModemSend

Syntax

```
void CofModemSend(char *send_string);
```

Description

Sends a string to the modem. Yields to other tasks while sending.

Parameters

`send_string`

A null terminated string to be sent to the modem.

Return value

None.

Library

`EXTERNAL_MODEM.LIB`

CofPPPshutdown

Syntax

```
int CofPPPshutdown(unsigned long timeout);
```

Description

Sends a Link Terminate Request packet. Waits for the link to be torn down.

Parameters

`timeout`

Number of milliseconds to wait before giving up on a response from the peer. Yields to other tasks while waiting.

Return value

1 - shutdown succeeded

0 - shutdown timed out

Library

`DCSPPP.LIB`

`CofPPPstart`

Syntax

```
int CofPPPstart(unsigned long timeout, int retry);
```

Description

Starts link negotiation process with a connected peer. Yields to other tasks.

Parameters

`timeout`

The number of milliseconds to wait between phases of negotiation before starting over.

`retry`

Number of times to retry the connection

Return value

1 - Negotiation succeeded;
0 - A link could not be negotiated.

Library

`DCSPPP.LIB`

`ModemClose`

Syntax

```
void ModemClose();
```

Description

Closes the serial driver down

Parameters

None.

Return value

None.

Library

`EXTERNAL_MODEM.LIB`

ModemConnected

Syntax

```
int ModemConnected();
```

Description

Returns true if the DCD line is asserted, meaning the modem is connected to a remote carrier.

Parameters

None.

Return value

1 - DCD line is active
0 - DCD inactive (nothing connected)

Library

`EXTERNAL_MODEM.LIB`

ModemExpect

Syntax

```
int ModemExpect(char *send_string, unsigned long timeout);
```

Description

Listens for a specific string to be sent by the modem.

Parameters

`send_string`

A null-terminated string to listen for.

`timeout`

Maximum wait in milliseconds for a character

Return value

1 if the expected string was received
0 if a timeout occured before receiving the string

Library

`EXTERNAL_MODEM.LIB`

ModemHangup

Syntax

```
int ModemHangup();
```

Description

Sends "ATH" and "ATZ" commands

Parameters

None.

Return value

1 - success

0 - modem not responding

Library

```
EXTERNAL_MODEM.LIB
```

ModemInit

Syntax

```
int ModemInit();
```

Description

Resets modem with AT, ATZ commands.

Parameters

None.

Return value

1 - success

0 - modem not responding

Library

```
EXTERNAL_MODEM.LIB
```

`ModemOpen`

Syntax

```
int ModemOpen(unsigned long baud);
```

Description

Starts up communication with an external modem.

Parameters

`baud`

The baud rate for communicating with the modem.

Return value

1 - External modem detected

0 - not connected to external modem

Library

`EXTERNAL_MODEM.LIB`

`ModemReady`

Syntax

```
int ModemReady();
```

Description

Returns true if the DSR line is asserted.

Parameters

None.

Return value

1 - DSR line is active

0 - DSR inactive (nothing connected)

Library

`EXTERNAL_MODEM.LIB`

ModemRinging

Syntax

```
int ModemRinging();
```

Description

Returns true if the RI line is asserted, meaning that the line is ringing.

Parameters

None.

Return value

1 - RI line is active

0 - RI inactive (nothing connected)

Library

`EXTERNAL_MODEM.LIB`

ModemSend

Syntax

```
void ModemSend(char *send_string);
```

Description

Sends a string to the modem.

Parameters

`send_string`

A null-terminated string to be sent to the modem.

Return value

None.

Library

`EXTERNAL_MODEM.LIB`

ModemStartPPP

Syntax

```
void ModemStartPPP();
```

Description

Hands control of the serial line over to the PPP driver.

Parameters

None.

Return value

None.

Library

`EXTERNAL_MODEM.LIB`

PPPclose

Syntax

```
void PPPclose();
```

Description

Closes the serial port and unloads the PPP interrupt service routine.

Parameters

None.

Return value

None.

Library

`PPPLINK.LIB`

`PPPinit`

Syntax

```
void PPPinit(unsigned long baud)
```

Description

Initializes the PPP driver, sets parameters.

Must be called immediately following a call to **`sock_init()`**.

Parameters

`baud`

The baud rate of the serial port PPP is running on (currently port C.)

Return value

None

Library

`DCSPPP.LIB`

`PPPflowcontrolOff`

Syntax

```
void PPPflowcontrolOff()
```

Description

Deactivates hardware flow control for the serial link.

Parameters

None.

Return value

None.

Library

`PPPLINK.LIB`

PPPflowcontrolOn

Syntax

```
void PPPflowcontrolOn()
```

Description

Activates hardware flow control for the serial link. The pins used for flow control are defined in PPPLINK.LIB as follows:

```
PPP_CTSPORT - the port address for the CTS input line
PPP_CTSPIN - the pin number of the CTS input line
PPP_RTSPORT - the port address of the RTS output line
PPP_RTSSHADOW -the name of the port's shadow register
PPP_RTSPIN - the pin number of the RTS output line
```

Parameters

None.

Return value

None.

Library

`PPPLINK.LIB`

PPPstart

Syntax

```
int PPPstart(unsigned long timeout, int retry);
```

Description

Starts link negotiation process with a connected peer

Parameters

`timeout`

Number of milliseconds to wait between phases of negotiation before starting over.

`retry`

Number of times to retry the connection.

Return value

1 - Negotiation succeeded;
0 - A link could not be negotiated.

Library

`DCSPPP.LIB`

PPPnegotiateIP

Syntax

```
void PPPnegotiateIP(unsigned long local_ip, unsigned long
remote_ip);
```

Description

Sets PPP driver to negotiate IP addresses for itself and the remote peer. Otherwise, the system will rely on the remote peer to set addresses.

Parameters

`local_ip`

IP number to use for this PPP connection.

`remote_ip`

IP number that the remote peer should be set to.

Return value

None.

Library

`DCSPPP.LIB`

PPPnegotiateDNS

Syntax

```
void PPPnegotiateDNS(unsigned long dns_ip);
```

Description

Sets PPP driver to configure a DNS address for the remote peer.

Parameters

`dns_ip`

IP number for the DNS server

Return value

None.

Library

`DCSPPP.LIB`

`PPPsetAuthenticator`

Syntax

```
void PPPsetAuthenticator(char *username, char *password);
```

Description

Sets the driver up to send a PAP authentication message to a peer when requested.

Parameters

`username`

The username to send to the peer. The argument string is not copied, so the argument string must stay constant.

`password`

The password to send to the peer. The argument string is not copied, so the argument string must stay constant

Return value

None.

Library

`DCSPPP.LIB`

PPPsetAuthenticatee

Syntax

```
void PPPsetAuthenticatee(char *username, char *password);
```

Description

Sets the driver up to send a PAP authentication message to a peer when requested.

Parameters

`username`

The username to send to the peer. The argument string is not copied, so the argument string must stay constant.

`password`

The password to send to the peer. The argument string is not copied, so the argument string must stay constant

Return value

None.

Library

`DCSPPP.LIB`

PPPshutdown

Syntax

```
int PPPshutdown(unsigned long timeout);
```

Description

Sends a Link Terminate Request packet. Waits for link to be torn down.

Parameters

`timeout`

Number of milliseconds to wait before giving up on a response from the peer.

Return value

1 - shutdown succeeded
0 - shutdown timed out.

Library

`DCSPPP.LIB`

`ResetPPP`

Syntax

```
void ResetPPP( );
```

Description

Under normal operations, this function will not be needed; the modem control functions make it unnecessary. There are, however, conditions that may make it useful.

Parameters

None.

Return value

None.

Library

```
DCSPPP.LIB
```