Technical Support Document
For
AT&T Paradyne
KeepInTouch Card™

By: Don Clementi

Revision: 1.0
April 21, 1994

CONTENTS

1. Introduction

This document is intended to provide developers of PCMCIA device drivers and PC manufacturers with detailed integration information for the AT&T Paradyne PCMCIA Data/Fax Modem. The modem is called the "KeepInTouch Card"™ (KIT for short).

AT&T Paradyne recognizes the tough issues involved in getting PCMCIA compliant PC's and cards robustly integrated in the short timescales being followed by leading-edge companies. We will provide technical assistance to companies involved in integration involving the KIT modem in the following forms:

1. Proprietary Technical Documentation (At start of integration effort)

2. Access to engineering staff for technical discussions (Duration of integration effort)

The information presented here has originated from a number of sources including the PCMCIA Specification (rev 2.0) and an AT&T Paradyne produced document entitled "3700 PCMCIA Interface Specification" (rev 1.4a). This document assumes a good working knowledge of these documents.

In order to promote the KIT modem on all major PC platforms, we have written a set of stand-alone utilities that provide modem configuration, test and upgrade functions. In more detail these utilities are:

- MCP - Modem Configuration Program
  Initializes the modem as an I/O card and configures the PCMCIA controller (PCIC) in the PC for modem operation on the specified COM port.

- MDP - Modem Diagnostic Program
  Provides a set of tests that exercise functional blocks of the modem reporting test status to the user.

- MUP - Modem Upgrade Program
  Implements a modem upgrade feature for the KIT modem, so that new firmware releases can be re-programmed into the modem using any PCMCIA compatible PC.

The MDP and MUP have been designed with the aim of being end-user utilities that a PC manufacturer could distribute with the KIT modem. The MCP was designed to provide a "stepping-stone" solution to the task of integrating the KIT modem with a PC's PCMCIA device driver. Each of these utilities currently work with the following PCMCIA Interface Controllers:

- INTEL PCIC 82365SL

- INTEL PHIC

- Toshiba Controller Gate Array

2. Overview of KIT Utilities

2.1. Modem Configuration Program

The MCP is a "stepping-stone" solution to the PCMCIA task of configuring the KIT modem for use as an I/O card when it has been inserted into the PCMCIA slot of the PC. It has been designed as a stand-alone program which the user executes after the KIT modem has been inserted, and before they attempt to communicate with the modem.

The MCP is *not* a TSR and so does not detect hot-insertion/removal events. The MCP currently does not contain a CIS parser, it assumes that the card in the designated PCMCIA slot is the KIT modem, and attempts to configure it as such. The MCP will not work with other manufacturer's PCMCIA cards. The MCP should **not** be used in combination with PC specific card services or driver services, as conflict for device control is likely to occur.

Through the command line interface, the MCP allows the user to configure the KIT modem on any one of the four standard COM ports. The modem power management modes can also be controlled once the card has been configured. The same is true for the audio output from the card, which can be enabled or disabled at anytime after the modem has been configured on a COM port.

The typical usage pattern of the MCP is outlined below:

1. KIT modem is inserted into PC
2. User configures modem for use (eg. MCP com2)
3. User uses a communications application
4. User commands modem into low power mode (eg. MCP sleep)
5. User commands modem back to full power mode (eg. MCP awake)
6. User uses a communications application
7. User removes KIT modem.

As you can see from the example session above, the MCP must **always** be used to configure the card for operation before any of the modem features become available. At any time after the initial configuration, the MCP can be used to control the KIT modem.

Please see the user guide in the README file on the distribution diskette for full instructions on using this utility.

2.2. Modem Diagnostic Program

The MDP has several operation modes, each of which provide access to a suite of functional modem tests. These tests have been designed to help identify any problems with the KIT modem, and to assist in the isolation of such problems.

This utility has been designed with the intention of being used by the end-user. It produces a log file of test results as well as having an interactive user-interface.

Please see the user guide in the README file on the distribution diskette for full instructions on using this utility.

2.3. Modem Upgrade Program

Another major feature of the KIT modem is its ability to be re-programmed with new firmware as new features are added to the product. The upgrade process requires access to the MUP and a Modem Firmware Release file.

The MUP can re-program the KIT modem with new modem firmware in approximately 30 seconds, and is very easy to use... You just type "MUP file=<release filename>" in most cases.

Please see the user guide in the README file on the distribution diskette for full instructions on using this utility.

3. PCMCIA Card Support Requirements

The PCMCIA standard specifies several different types of PCMCIA cards, each of which can be hot-inserted/removed (ie. while PC is powered-up). The standard also states that all cards shall present themselves as memory cards before they have been appropriately identified and subsequently configured for use.

In order to write a generalized PCMCIA driver for a range of cards, the PC must have a method of identifying the card which has been inserted. This is acheived by the implementation of a parser which scans the Card Information Structure (CIS) on the card. Although the CIS can become quite complex, it is important that a good parser be implemented in order to support the "plug and play" advertisement that PCMCIA is sending loud and clear to the user community.

Once the parser has identified all of the relevant information in a card's CIS, it must then act on that information to control the PC and the card in a manner consistant with the card's CIS-declared interface and feature set.

The PCMCIA driver must also include a method for controlling the power setting of the card, selecting the lowest available power mode when the card is not in use. The ultimate goal is to make power management of PCMCIA cards invisible to the user.

The next few sections cover the aspects of the CIS, card configuration and control of card power modes, as these subjects relate to the KIT modem.

4. Card Information Structure

4.1. Purpose

The CIS is a linked list of data objects (called tuples) that describe the card's manufacturer, revision information, type, feature set, power requirements, etc. When correctly parsed, the CIS yields significant information about the card, allowing the PCMCIA driver software to control the PC and the card in a sensible manner. It is vitally important that a good "generic" parser be implemented, and that it can handle a variety of card types, reliably.

4.2. Minimum Parsing Recommendations

In order to be useful, the PCMCIA driver must perform some parsing of the CIS when a card is inserted. Ultimately, we strongly recommend the implementation of a full, generic CIS parser in any commercial-strength PCMCIA driver, but in the short term, we recognize that the CIS can become quite complex and that a short-term solution maybe required. To support this, we are describing those tuples that we believe are important to such a short-term driver solution, from the perspective of the KIT modem.

The CIS on the KIT modem starts at the base address of the memory window that has been setup in the PC's PCMCIA controller. The MCP defaults this memory window base address to D0000h in the PC. The window needs to be 2k-bytes (or greater) in size. Note that the PC must access the CIS bytes by generating even addresses only, since the KIT modem is an 8-bit card. This is a PCMCIA requirement.

The parser must examine the first byte of each tuple in the CIS to determine the tuple type. The following tuples may be ignored by the parser (*in the short term solution*) jumping to the start of the next tuple by following the offset in the byte that follows the tuple type (link field). (The symbolic names used in the following section are derived from the PCMCIA V2.0 Specification, pages xx - yy).

No Processing Necessary (*in the short term solution*)

- CISTPL_DEVICE
- CISTPL_DEVICE_A
- CISTPL_FUNCID
- CISTPL_FUNCE (all four of them)
- CISTPL_CE (all four of them, but see description of CISTPL_CONF below)

When the parser reaches the tuple with a type code of CISPTPL_END, it has finished parsing.

Now let's take a look at those tuples that do need to be processed in the interests of recognizing the KIT modem today.

Processing Required

CISTPL_VER_1
This tuple can be the hook to determine the card manufacturer, the product name and the version of the firmware in the KIT modem. Going forward from firmware version C01.20.00, we do not anticipate any changes to the PCMCIA interface which would affect the processing in the PCMCIA driver on the PC. Note that when the MUP is used to change the modem firmware, this firmware version number will be changed. As a result, recognition of a KIT card should **not** be tied to a particular firmware revision level.

CISTPL_MANFID
This tuple uniquely defines the card manufacturer using a PCMCIA-allocated number. AT&T Paradyne's manufacturer ID # is: 0x100 (256_{10}). We recommend that you use this tuple to identify AT&T Paradyne as opposed to the CISTPL_VER_1 tuple string.

CISTPL_CONF
The KIT modem supports PCMCIA registers; COR, CCSR and PRR, as described in the TPCC_RMSK field at tuple offset 12 (as viewed from PC). Since different cards place these registers at a different offset from the start of the memory window, it is important to determine that offset by examining the TPCC_RADR field (maybe multiple bytes as in KIT modem) within this tuple.

It is not essential to process the configuration entry tuples that are referenced in the CISTPL_CONF tuple, once the device driver recognizes that the KIT card is a modem, that can be configured on any of the COM ports. The one exception that may contradict this is that during card initialization, the PCMCIA driver needs to determine what value it needs to write into the COR in order to configure the KIT modem correctly. In the KIT modem, those COR values are:

COM1 = 20h COM2 = 21h COM3 = 22h COM4 = 23h

However, in other cards, these configuration values will most likely be different. The CIS parser can determine these values by examining the TPCE_INDX field of the four CISTPL_CE tuples. A sub-element of that field specifies the value that needs to be written to the COR during card configuration.

The complex issue of power requirements is quite simple at present. The KIT modem requires only a Vcc of 5V from the PC. Therefore, all of the power descriptors in the first CISTPL_CE tuple can be ignored if the PCMCIA driver wishes to hard-code this power setting for the KIT modem.

5. Card Configuration

After insertion, the KIT modem must be configured as a modem by the host PC before it will respond to communications applications. This can be acheived by waiting for the card to indicate READY on pin 16 of the PCMCIA interface, and then writing the COR on the card with a value dependant upon the COM port you wish to use.

COM1: COR = 0x20, COM2: COR = 0x21, COM3: COR = 0x22, COM4: COR = 0x23

PLEASE NOTE... you should not attempt to access the CIS on the KIT card until it has indicated READY on pin 16 of the PCMCIA interface. This may take up to one second from initial power-up.

When the PC writes to the COR for the first time, the KIT modem performs some initialization for approximately 3msecs, during which time the PC should not write to the card, or repeatedly poll the PCMCIA registers with no delay between each read (we recommend at least a 15msec delay in polling loops). When the card is ready for use as a modem, it indicates READY in bit Rdy/Bsy of the PRR. At the same time, it also sets the CRdy/Bsy bit in the PRR, which acts like a status latch. The modem will also set the Changed bit in the CCSR and activate the STSCHG signal on pin 63 if it is enabled. The host PC **MUST RESET** the CRdy/Bsy bit in the PRR to indicate that it has seen the change in status. This reset action causes the card to place the STSCHG signal on pin 63 of the interface back in its inactive state.

The KIT modem is now ready for use as a modem.

5.1. PC Driver Source Code

To assist in the implementation of the requirements in the previous section, we present some 'C' code that demonstrates the card configuration actions. This code works with INTEL's PCIC, the 82365SL. Just one comment is necessary, the 'ReadGA' and 'UpdateGA' functions, read and write to the PCIC register in the first argument. In the case of 'UpdateGA', the second argument is the value written to the register.

```
int ConfigureKITmodem()
{
        int memoryWinStart = 0xd000;            /* Base address of memory window */
        int memoryWinEnd   = 0xdfff;            /* 4k memory window is smallest available */

        /*** ENSURE A CARD IS IN THE FIRST SLOT ***/
        if (!(ReadGA(0x01) & 0x0C))
        {
                        printf("No PC Card Detected!!\n");
                exit(ERR_NO_CARD);
        }

        /*** POWER ON SLOT & ENABLE SLOT INTERFACE ***/
          UpdateGA(0x02, 0x10);                         /* 5V Vcc On to slot */
          delay(100);                           /* delay approx 100 ms */
          UpdateGA(0x02, 0x90);                         /* Enable Interface */
          delay(10);

          /*** RESET CARD & INITIALIZE AS A MEMORY CARD ****/
          UpdateGA(0x03, 0x00);                         /* RESET Card */
          delay(200);
          UpdateGA(0x03, 0x40);                         /* Out of RESET, Memory Card
interface */
          delay(10);

        /*** SETUP MEMORY WINDOW 0 ***/
          UpdateGA( 0x11, 0x00);                /* Memory Window Start */
         UpdateGA( 0x10, (Uchar)((memoryWinStart & 0xFF00)>>8));
          UpdateGA( 0x13, 0x00);                /* Memory Window End */
         UpdateGA( 0x12, (Uchar)((memoryWinEnd   & 0xFF00)>>8));
          temp = ~memoryWinStart + 1;
         UpdateGA( 0x14, (Uchar)((temp & 0xFF00)>>8));          /* Offset of window to card
memory */
          UpdateGA( 0x15, 0x7F);
          UpdateGA(0x06, 0x01);                         /* Enable Memory Window 0 */

        /*** WAIT UNTIL CARD INDICATES READY ON PIN 16 OF PCMCIA INTERFACE ***/
                while (! (ReadGA(0x01) & 0x20))
          {
              delay(15);                                /* 15ms delay - approximate */
          }

        /*** NOW CONFIGURE CARD AS A MODEM DEVICE ***/
        if (comPort == COM1)
        {
                UpdateGA(0x03, 0x64);           /* IRQ4 selected */
                UpdateGA(0x09, 0x03);           /* MSB of I/O Start Addr */
                UpdateGA(0x08, 0xF8);           /* LSB of I/O Start Addr */
                UpdateGA(0x0B, 0x03);           /* MSB of I/O Stop Addr */
```

```
                UpdateGA(0x0A, 0xFF);                   /* LSB of I/O Stop Addr */
        }
        else if (comPort == COM2)
        {
                UpdateGA(0x03, 0x63);                   /* IRQ3 selected */
                UpdateGA(0x09, 0x02);                   /* MSB of I/O Start Addr */
                UpdateGA(0x08, 0xF8);                   /* LSB of I/O Start Addr */
                UpdateGA(0x0B, 0x02);                   /* MSB of I/O Stop Addr */
                UpdateGA(0x0A, 0xFF);                   /* LSB of I/O Stop Addr */
        }

          UpdateGA(0x07, 0x00);                                 /* I/O Control */
          UpdateGA(0x06, 0x41);                         /* Enable I/O Window

        delay(100);
        cor &= ~0x3F;                                           /* Clear old config bits
        cor |= comPort -1 + 0x20;                               /* Setup for I/O card */
        UpdateCOR(cor);                                         /* Write new COR to modem card
*/

        /*** NOW WAIT FOR THE MODEM TO INDICATE READY IN PRR ***/
         for (i=0; i<TIMEOUT; i++)
        {
                delay(15);
                prr = ReadPRR();
                  if ((prr & CRDYBSY) && (prr != 0xff))         /* All ff's means no card access
possible */
                {
                        if (prr & RRDYBSY)              /* IF RRdy/Bsy bit indicates READY */
                     {
                         ClearBit(CRDYBSY);             /* Clear CRdy/Bsy bit */
                            return CONFIGURED;                  /* We're done */
                         }
                     else                               /* ELSE RRdy/Bsy bit indicates BUSY */
                         {
                            ClearBit(CRDYBSY);          /* Clear CRdy/Bsy bit and continue */
                     }
                         }
        }
        return TIMEOUT;                                 /* If we timed-out return this status */
}
```

6. Power Management Control

6.1. Power Down Mode

This mode is entered when the PwrDwn bit in the CCSR is set by the host PC. From the time the PC sets the PwrDwn bit, the KIT modem requires less than 3msecs to fully enter the power down mode. During that time it is strongly recommended that the PC not write to the card.

This is all that is involved to access the PCMCIA defined power down modem of the KIT modem.

6.2. Return To Full Power Mode - Commanded

Returning the KIT modem to full power mode, from PCMCIA power down mode is a little more involved, and has a lot in common with the initial modem configuration sequence mentioned earlier in this document.

The host PC can command the KIT modem back to full power mode by resetting the PwrDwn bit in the CCSR and then waiting for the card to indicate BUSY in the RRdy/Bsy bit in the PRR (This takes approximately one second). The wait is due to the card's circuitry being re-powered and the modem firmware going through a re-initialization phase.

The PC MUST watch the KIT modem transition from READY to BUSY and then from BUSY to READY, (using bit RRdy/Bsy in the PRR) before it considers the modem to be awake. In more detail, when the modem indicates BUSY in the PRR, it will also set the CRdy/Bsy bit in the PRR and set the Changed bit and activate the STSCHG signal on pin 63 (if enabled). The host PC response MUST BE to RESET the CRdy/Bsy bit in the PRR to indicate that it has seen the transition to BUSY. A short time later, the card will transition from BUSY to READY, setting all of the other bits and the STSCHG signal as just described. The host PC response MUST BE be to RESET the CRdy/Bsy bit in the PRR.

The modem is now in full power mode and ready to accept AT commands from a communications application. It is possible that you may experience difficulty waking the card up if you do not follow the above sequencing and timing guidelines. If the wake-up sequence is not working, suspect timing first.

6.3. Return To Full Power Mode - Ring Detected

When the host PC has placed the KIT modem in its PCMCIA defined, power down mode the card can also return to full power on the occurance of RING.

When the first RING is detected, the modem is re-powered. As the modem initializes it generates the changes to the PRR and STSCHG just like in section 6.2. The PC software MUST follow the same protocol as in section 6.2 in order to successfully use the modem once it has come out of sleep mode.

We recognize that the awake from sleep mode on RING detect is not a PCMCIA standard at this time, and are actively working with other modem vendors to ensure that a long-term solution appears in the not too distant future.