RS·232C Adapter

For use with VIC·20 and COMMODORE·64

VIC-1011A

commodore COMPUTER

RS·232C Adapter

For use with VIC·20 and COMMODORE·64

commodore COMPUTER

TABLE OF CONTENTS

RS-232C Interface Cartridge 1
Connecting The Cartridge 1
Using The VIC/C64's RS-232 Capabilities 1
Commodore Information Network 2
Getting Deeper Into The RS-232 Connection 2
RS-232 Interface Description –
Built - In Software 3
Opening An RS-232 Channel 3
Getting Data From RS-232 Channel 4
Sending Data To An RS-232 Channel 7
Closing An RS-232 Data Channel 8
Explanation of RS-232C Signal 12
VIC/C64 User Part Connector 12
RS-232C Cartridge Connector 13
Interface Handshake Sequence Of
X-Line Mode 14
Appendix A A-1
User Callable Kernal Routines A-2
Receiver/Transmitter Buffer Base
Location Pointers A-7
Zero-Page Memory Locations And
Usage For RS-232 System Interface A-8
Nonzero-Page Memory Locations And
Usage For RS-232 System Interface A-8

RS-232C INTERFACE CARTRIDGE

This RS-232C "TERMINAL-TYPE" CARTRIDGE allows you to use standard RS-232 devices with your VIC 20/C64, through the User Port. The cartridge conforms to EIA Standard (Aug. 1979) for RS-232 Interface.

CONNECTING THE CARTRIDGE

Following these steps to connect your RS-232 cartridge:

1. Turn OFF the VIC/C64 ON/OFF switch.

2. Connect an RS-232 cable between the RS-232 Interface Cartridge and RS-232 device (i.e. modem). (RS-232 Cables are sold by most computer and electronic stores.)

3. Insert the RS-232 Cartridge into the User Port slot in the back left-hand corner of the VIC/C64. The cartridge's metal label should be facing UP.

4. Turn ON the VIC-20/C64.

USING THE VIC/C64's RS-232 CAPABILITIES

If you're using an RS-232 modem with VIC/C64 you can now switch to the manual which comes with your modem and proceed...whether you're communicating with other VIC/C64's or accessing a telecomputing service.

COMMODORE INFORMATION NETWORK

Commodore has set up a special "Commodore Information Network" which CompuServe subscribers can access to get current product information and technical assistance, or exchange tips with other Commodore owners. Here are a few of the services provided through this network. The subscriber will be billed for his continued use of the services provided through this network:

1. Commodore Hotline
2. Commodore Newsletter
3. User Bulletin Board
4. Product Announcements
5. User Club Directory
6. Programming Tips
7. Technical Support
8. Commodore Graphics
9. *Program of the Month
10. *Public Domain Software

To get started, join the CompuServe service, hook up your modem, follow the instructions in the CompuServe manual, sign onto CompuServe and at the first prompt type:

GO CBM

Note: These features will be available at a later date. Announcements will be made as services are expanded.

GETTING DEEPER INTO THE RS-232 CONNECTION

The following exerpt from the VIC 20 PROGRAMMER'S REFERENCE GUIDE provides more technical information about the VIC/C64's RS-232 interface capability, including special RS-232 programming information.

RS-232 INTERFACE DESCRIPTION – BUILT-IN SOFTWARE

The RS-232 interface software can be accessed from BASIC or from the KERNAL for machine language programming. RS-232 on the BASIC level uses the normal BASIC commands: OPEN, CLOSE, CMD, INPUT#, GET#, PRINT#, and the reserved variable ST. INPUT# (CHRIN for the machine language programmers in the audience) and GET# (GETIN) fetch data from the receiving buffer, while PRINT# (chrout) and CMD place data into the transmitting buffer. The use of these commands (and examples) will be described more fully later.

The RS-232 KERNAL byte/bit level handlers run under the control of the 6522 device timers and interrupts. The 6522 generates NMI requests for RS-232 processing. This allows background RS-232 processing to take place during BASIC and machine language programs. There are built-in hold-offs in the KERNAL cassette and serial bus routines to prevent disruption of data storage/transmission by the NMI's generated by the RS-232 routines. During cassette or serial bus activities data cannot be received from RS-232 devices. Because these hold-offs are internal to VIC-20/C64 (assuming care is taken in programming) no interference should result.

There are two buffers in the VIC-20/C64 RS-232 interface to help prevent loss of data when transmitting or receiving RS-232. The VIC/C64 RS-232 KERNAL buffers consist of two first-in/first-out (FIFO) buffers, each 256 bytes long, at the top of memory. The opening of an RS-232 channel automatically allocates 512 bytes of memory for these buffers. If there is not enough free space beyond the end of your BASIC program no error message will be printed, and the end of your program will be destroyed. SO BE CAREFUL!

These buffers are automatically removed by the CLOSE command.

OPENING AN RS-232 CHANNEL

Only one RS-232 channel should be open at any time; a second OPEN statement will cause the buffer pointers to be reset. Any characters in either the transmit buffer or the received buffer will be lost.

Up to 4 characters may be sent in the filename field. The first two are the control and command register characters; the other two are reserved for future system options. Baud rate, parity, and other options can be selected through this feature.

No error-checking is done on the control word to detect a nonimplemented baud rate, so that any illegal control word will cause the system output to operate at a very slow rate (below 50 baud).

BASIC SYNTAX

OPENIf,2,0,"<control register><command register>"

If—Normal logical file id (1-255). If If >127 then linefeed follows carriage return.

<control register>—Single byte character (see Figure 1) (required to specify baud rate)

<command register>—Single byte character (see Figure 2) (this character is NOT required)

KERNAL ENTRY

OPEN ($FFCO) — See Appendix A for more information on entry and exit conditions.

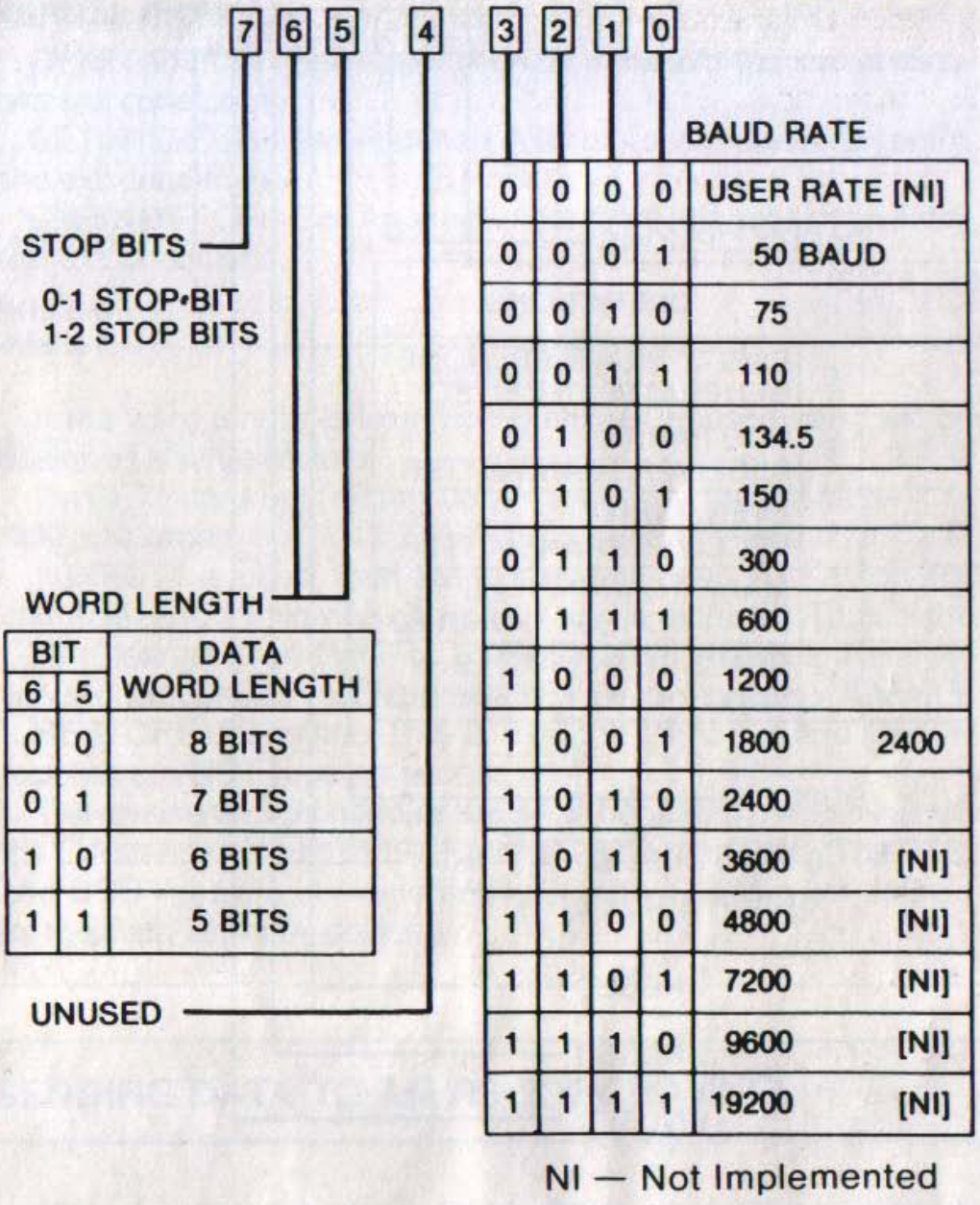
NOTE

IMPORTANT: In a BASIC program, the RS-232 OPEN command should be performed before using any variable or DIM statement, since an automatic CLR is performed when an RS-232 channel is OPENed (because of the allocation of 512 bytes at the top of memory.) Also remember that your program will be destroyed if 512 bytes of space are not available at the time of the OPEN statement.

GETTING DATA FROM RS-232 CHANNEL

When getting data, the VIC-20/C64 receiver buffer will hold 255 characters before a buffer overflow. This is indicated in the RS-232 status word (ST from BASIC, RSSTAT ($297) from machine language). If this occurs, all characters received during a full buffer condition are lost. Obviously it pays to keep the buffer as clear as possible.

If you wish to receive RS-232 data at high speeds (BASIC can only go so fast, especially considering string garbage collection. When BASIC takes the time to do garbage collection the receiver buffer can overflow because BASIC will not pay attention to RS-232 during collection), you will have to use machine language routines to handle the data bursts.

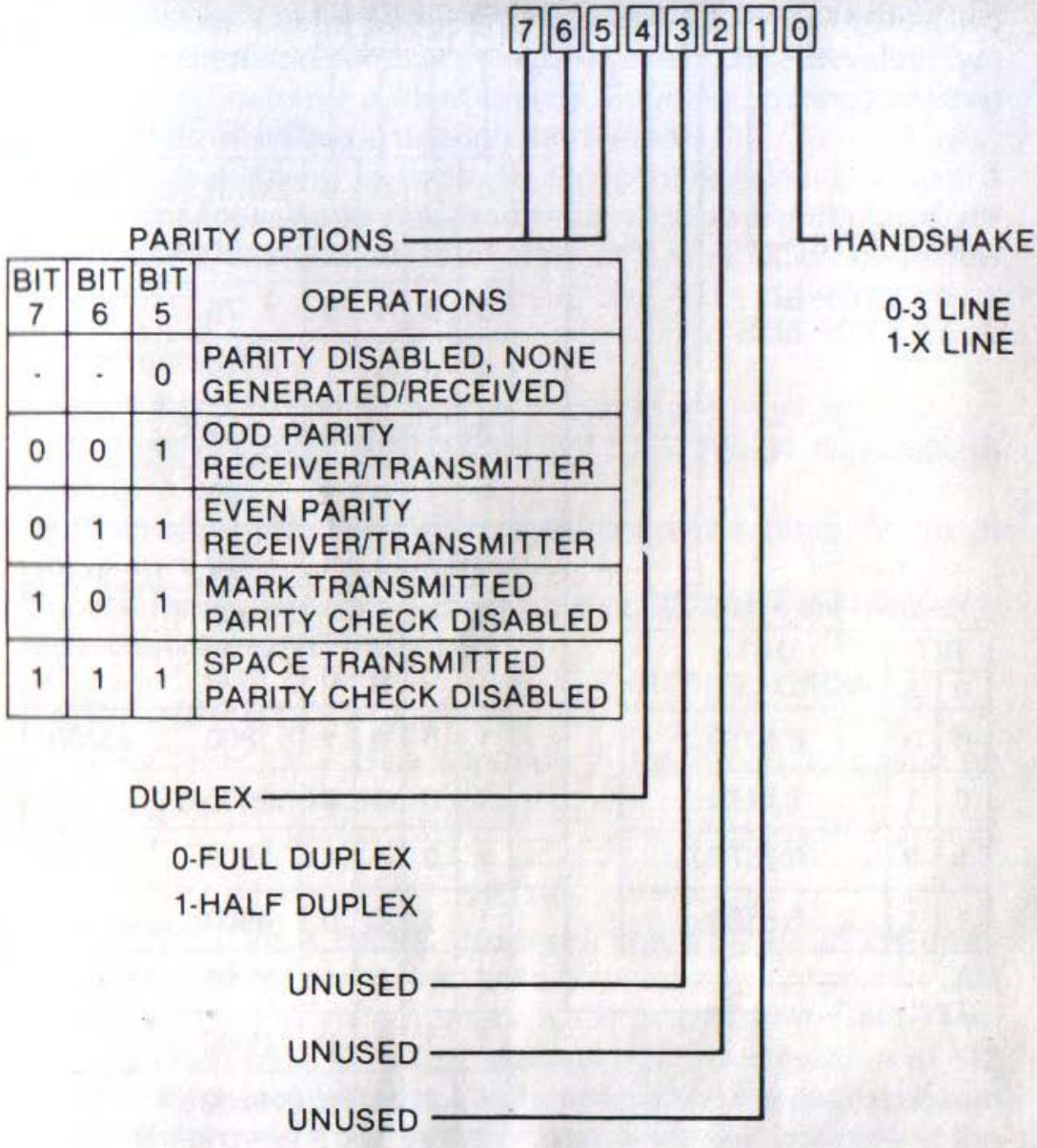


BIT 6	BIT 5	DATA WORD LENGTH
0	0	8 BITS
0	1	7 BITS
1	0	6 BITS
1	1	5 BITS

3	2	1	0		
0	0	0	0	USER RATE [NI]	
0	0	0	1	50 BAUD	
0	0	1	0	75	
0	0	1	1	110	
0	1	0	0	134.5	
0	1	0	1	150	
0	1	1	0	300	
0	1	1	1	600	
1	0	0	0	1200	
1	0	0	1	1800	2400
1	0	1	0	2400	
1	0	1	1	3600	[NI]
1	1	0	0	4800	[NI]
1	1	0	1	7200	[NI]
1	1	1	0	9600	[NI]
1	1	1	1	19200	[NI]

NI — Not Implemented

Figure 1. Control Register



BIT 7	BIT 6	BIT 5	OPERATIONS
-	-	0	PARITY DISABLED, NONE GENERATED/RECEIVED
0	0	1	ODD PARITY RECEIVER/TRANSMITTER
0	1	1	EVEN PARITY RECEIVER/TRANSMITTER
1	0	1	MARK TRANSMITTED PARITY CHECK DISABLED
1	1	1	SPACE TRANSMITTED PARITY CHECK DISABLED

Figure 2. Command Register

BASIC SYNTAX

Recommended: GET#If,<string variable>
NOT Recommended: INPUT#If,<variable list>

KERNAL ENTRY

CHKIN ($FFC6)–See Appendix A for more information on entry and exit conditions.

GETIN ($FFE4)–See Appendix A for more information on entry and exit conditions.

CHRIN ($FFCF)–See Appendix A for more information on entry and exit conditions.

NOTES

If the word length is less than 8 bits, all unused bit(s) will be assigned a value of zero.

If a GET# does not find any data in the buffer, the character"" (a null) is returned.

If INPUT# is used, then the system will hang until a non-null character and a following carriage return is received. Thus, if the CTS (Clear to Send Data) or DSR (Data Set Ready to Receive) line(s) disappear during character INPUT #, the system will hang in a RESTORE-only state. This is why the INPUT # and CHRIN routines are NOT recommended.

The routine CHKIN handles the x-line handshake which follows the EIA standard (August 1979) for RS-232C interfaces. (The RTS, and DCD lines are implemented with the VIC computer defined as the Data Terminal device.)

SENDING DATA TO AN RS-232 CHANNEL

When sending data, the output buffer can hold 256 characters before a full buffer hold-off occurs. The system will wait in the CHROUT routine until transmission is allowed or the RUN/STOP-RESTORE keys are used to recover the system through a WARM START.

BASIC SYNTAX

CMD lf—acts same as in BASIC specifications
(see VIC-20 User Reference Manual)
PRINT#lf,<variable list>

KERNAL ENTRIES

CHKOUT ($FFC9)–See Appendix A for more information on entry and exit conditions.

CHROUT ($FFD2)–See Appendix A for more information on entry and exit conditions.

NOTES

IMPORTANT: There is no carriage-return delay built into the output channel so a normal RS-232 printer cannot correctly print, unless some form of hold-off (asking the VIC-20/C64 to wait) or internal buffering is implemented by the printer. The hold-off can easily be implemented in your program. If a CTS (x-line) handshake is implemented, the VIC-20/C64 buffer will fill and hold off more output until transmission is allowed by the RS-232 device.

The routine CHKOUT handles the x-line handshake, which follows the EIA standard (August 1979) for RS-232-C interfaces. The RTS, and DCD lines are implemented with the VIC defined as the Data Terminal Device.

CLOSING AN RS-232 DATA CHANNEL

Closing an RS-232 file discards all data in the buffers at the time of execution (whether or not it had been transmitted or printed out), stops all RS-232 transmitting and receiving, sets the RTS and Sout lines high, and removes both RS-232 buffers.

BASIC SYNTAX

CLOSE lf

KERNAL ENTRY

CLOSE ($FFC3)–See Appendix A for more information on entry and exit conditions.

VIC-20 NOTE

Care should be taken to ensure all data is transmitted before closing the channel. A way to check this from BASIC is:

```
360 IF ST=0 AND (PEEK(37151) AND 64)=64: GOTO 360
370 CLOSE lf.
```

Table 1. VIC-20 USER – PORT LINES

(6522 DEVICE #1 loc $9110 - PORT B OUTPUT Register)

PIN ID	6522 ID	DESCRIPTION	EIA	ABV	IN/ OUT	MODES
C	PB0	RECEIVED DATA	(BB)	Sin	IN	1 2
D	PB1	REQUEST TO SEND	(CA)	RTS	OUT	1*2
E	PB2	DATA TERMINAL READY	(CD)	DTR	OUT	1*2
F	PB3	RING INDICATOR	(CE)	RI	IN	3
H	PB4	RECEIVED LINE SIGNAL	(CF)	DCD	IN	2
J	PB5	UNASSIGNED	( )	XXX	IN	3
K	PB6	CLEAR TO SEND*	(CB)	CTS	IN	2
L	PB7	DATA SET READY	(CC)	DSR	IN	2
B	CB1	RECEIVED DATA	(BB)	Sin	IN	1 2
M	CB2	TRANSMITTED DATA	(BA)	Sout	OUT	1 2
A	GND	PROTECTIVE GROUND	(AA)	GND		1 2
N	GND	SIGNAL GROUND	(AB)	GND		1 2 3

MODES

1)—3-LINE INTERFACE (Sin,Sout,GND)
2)—X-LINE INTERFACE
3)—USER AVAILABLE ONLY (Unused/unimplemented in code.)
*—These lines are held high during 3-LINE mode.

*Note: The CLEAR TO SEND (CTS) signal used with the X-line interface is not implemented. It is possible to read the CTS signal directly from the 6522 chip with a machine language program.

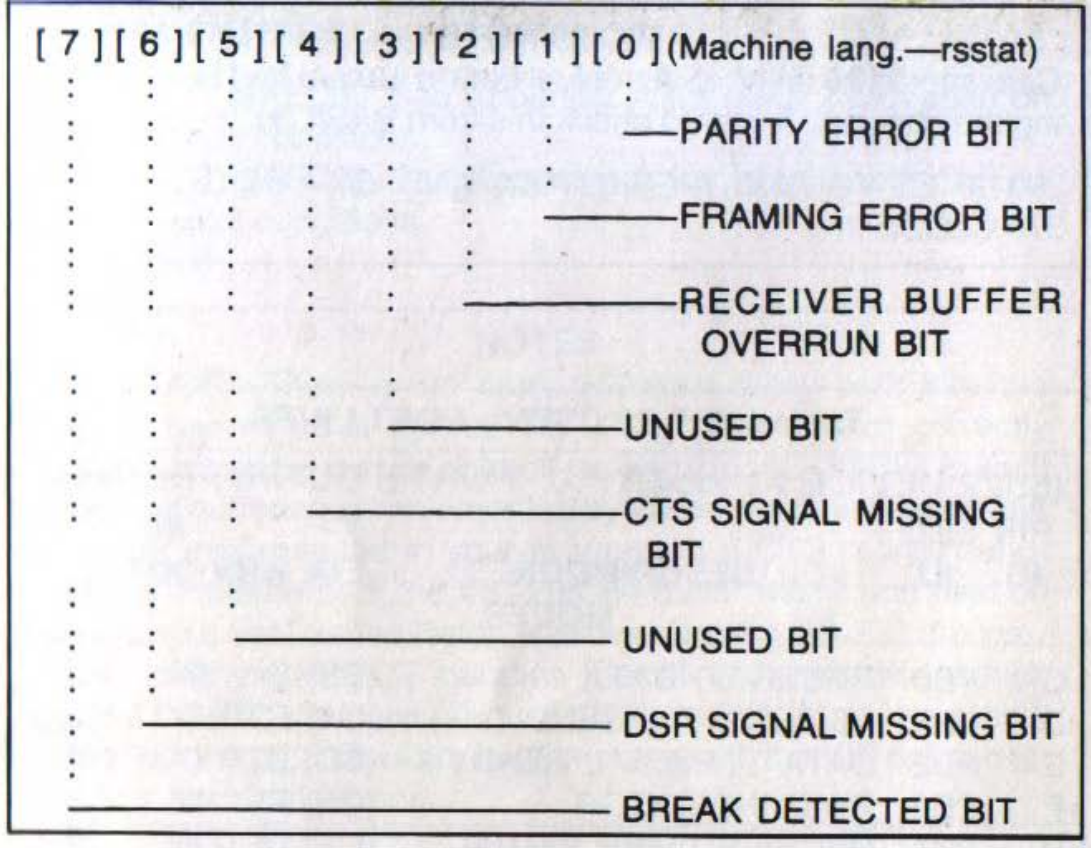


Figure 3. RS-232 Status Register

Status register can be accessed in BASIC by the command ST and in machine code register $0297 can be read for the status.

NOTES

If the BIT = 0, then no error has been detected.

The RS-232 status register can be read from BASIC using the variable ST.

If ST is read by BASIC or by using the KERNAL READST routine the RS-232 status word is cleared upon exit. If multiple uses of the STATUS word are necessary the ST should be assigned to another variable, i.e.

```
SR = ST: REM ASSIGNS ST TO SR
```

The RS-232 status is read (and cleared) only when the RS-232 channel was the last external I/O used.

C64 NOTE

Care should be taken to ensure all data is transmitted before closing the channel. A way to check this from BASIC is:

```
100 SS = ST: IF(SS = 0 OR SS = 8) THEN 100
110 CLOSE lfn
```

Table 2. C64 USER-PORT LINES

(6526 DEVICE #2 loc $DD00-$DD0F)

PIN ID	6526 ID	DESCRIPTION	EIA	ABV	IN/ OUT	MODES
C	PB0	Received Data	(BB)	Sin	IN	1 2
D	PB1	Request to Send	(CA)	RTS	OUT	1*2
E	PB2	Data Terminal Ready	(CD)	DTR	OUT	1*2
F	PB3	Ring Indicator	(CE)	RI	IN	3
H	PB4	Received Line Signal	(CF)	DCD	IN	2
J	PB5	Unassigned	( )	XXX	IN	3
K	PB6	Clear to Send	(CB)	CTS	IN	2
L	PB7	Data Set Ready	(CC)	DSR	IN	2
B	FLAG2	Received Data	(BB)	Sin	IN	1 2
M	PA2	Transmitted Data	(BA)	Sout	OUT	1 2
A	GND	Protective Ground	(AA)	GND		1 2
N	GND	Signal Ground	(AB)	GND		1 2 3

MODES:

1) – 3-LINE INTERFACE (Sin, Sout, GND)
2) – X-LINE INTERFACE
3) – USER AVAILABLE ONLY (Unused/unimplemented in code.)

* – These lines are held high during 3-LINE mode.

SIMPLE BASIC PROGRAM FOR VIC-20

```
  10 REM THIS PROGRAM SENDS AND RECEIVES DATA
TO/FROM A SILENT 700 TERMINAL MODIFIED FOR PET
ASCII
  20 REM TI SILENT 700 SET-UP: 300 BAUD, 7-BIT ASCII,
MARK PARITY, FULL DUPLEX
  30 REM SAME SET-UP AT COMPUTER USING 3-LINE
INTERFACE
100 OPEN 2,2,3,CHR$ (6 + 32) + CHR$ (32 + 128): REM OPEN
THE CHANNEL
110 GET#2,A$: REM TURN ON THE RECEIVER CHANNEL
(TOSS A NULL)
200 REM MAIN LOOP
210 GET B$: REM GET FROM COMPUTER KEYBOARD
220 IF B$<>"" THEN PRINT#,B$;: REM IF A KEY
PRESSED, SEND TO TERMINAL
230 GET#2,C$: REM GET A KEY FROM THE TERMINAL
240 PRINT B$;C$;: REM PRINT ALL INPUTS TO THE
COMPUTER SCREEN
250 SR = ST: IF SR = 0 THEN 200: REM CHECK STATUS, IF
GOOD THEN CONTINUE
```

```
300 REM ERROR REPORTING
310 PRINT "ERROR: ";
320 IF SR AND 1 THEN PRINT "PARITY"
330 IF SR AND 2 THEN PRINT "FRAME"
340 IF SR AND 4 THEN PRINT "RECEIVER BUFFER FULL"
350 IF SR AND 128 THEN PRINT "BREAK"
360 IF ST = 0 AND (PEEK(37151) AND 64) = 64 THEN 360: REM
WAIT UNTIL ALL CHARACTERS ARE TRANSMITTED
370 CLOSE 2: END
```

Note for C64– Change line 360 to read:

```
360 SS = ST: IF (SS = 0 OR SS = 8) THEN 360
```

EXPLANATION OF RS-232C SIGNAL

The signal in VIC/C64 and that of the user port are at 0V∼+5V TTL level. These signals are converted into those at −9V ± 3V ∼ + 9V ± 3V level by RS-232 C adapter.

VIC/C64 USER PORT CONNECTOR

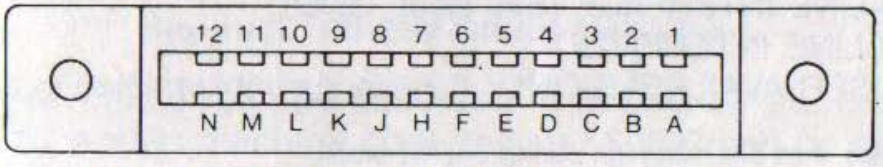


FIGURE 4. External View of VIC/C64 User Port Connector on RS232 Connector

Terminal No.	Signal	Explanation	Direction Of Signal	Mode
A	GND	Ground		1 2
B	S in	Received Data	Input	1 2
C	S in	Received Data	Input	1 2
D	RTS	Request to Send	Output	1* 2
E	DTR	Data Terminal Ready	Output	1* 2
F	(NC)	No Connection		
H	DCD in	Detect Carrier	Input	2
J	DCD out	Detection Carrier	Output	**
K	CTS	Clear To Send	Input	
L	DSR	Data Set Ready	Input	2
M	S out	Transmitted Data	Output	1 2
N	GND	Signal Ground		
1	GND	Ground		
2	Vcc	+5V		
3 thru 9	NC	No Connection		
10	AC9V	AC9V (***)		
11	AC94	AC94		
12	GND	Signal Ground		

Mode:
1 . . . . 3 line (S in, S out, GND).
2 . . . . X line

(Note) * These lines will be kept HIGH during 3-line mode
** Jumper is not connected.
*** For only VIC 1011B CURRENT LOOP TYPE. A limited number of RS-232 "Current Loop" cartridges are available, by special order.

RS-232C CARTRIDGE CONNECTOR

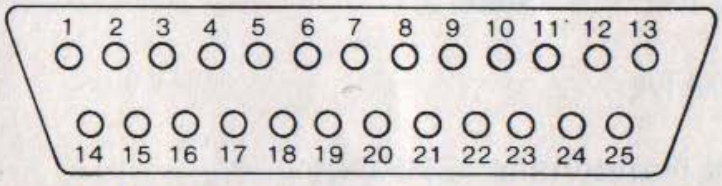


Figure 5. RS-232C Interface Cartridge Connector, Facing The Cable

PIN NO.	SIGNAL	EXPLANATION	EIA	SIGNAL DIRECTION	MODE
1	GND	Ground			1 2
2	SD	Transmitted Data	BA	Output	1 2
3	RD	Received Data	BA	Input	1 2
4	RS	Request To Send	CA	Output	1* 2
5	CS	Transmission Is Possible	CB	Input	2
6	DR	Data Set Ready	CC	Output	
7	GND	Signal Ground			
8	CD	Carrier Detect	CF	Output	2
9	CL+	Current Loop+			***
10	CL−	Current Loop−			***
11-19		No Connection			
20	ER	Data Terminal Ready	CD	Input	1*
21-25		No Connection			

INTERFACE HANDSHAKE SEQUENCE OF X-LINE MODE

```
DATA OUT Handshake (CHKOUT)
IF (DSR) = LOW THEN <DSR ERROR & EXIT>
IF (RTS) = LOW
 [LOOP:UNTIL (ALL IMMEDIATE CHARS INPUT)
  LOOP: UNTIL (CTS) = LOW
  (RTS): = HIGH
  LOOP
    IF (DSR) = LOW THEN <DSR ERROR & EXIT>
  UNTIL (CTS) = HIGH
 ]
SEND DATA

DATA IN Handshake (CHKIN)
IF (DSR) = LOW THEN <DSR ERROR FLAG & EXIT>
IF (RTS) = HIGH THEN
 [LOOP:UNTIL (ALL IMMEDIATE CHARS SENT)
  (RTS): = LOW
  LOOP:UNTIL (DCD) = HIGH
 ]
WAIT FOR INPUT DATA
```

APPENDIX A

Function name: CHKIN

Purpose: Open a channel for input
Call address: $FFC6
Communication registers: .X
Preparatory routines: (OPEN)
Error returns: 3,5,6
Stack requirements: None
Registers affected: .A, .X

Description: Any logical file that has already been opened by the KERNAL OPEN routine can be defined as an input channel by this routine. Naturally, the device on the channel must be an input device. Otherwise, an error will occur, and the routine will abort.

If you are getting data from anywhere other than the keyboard, this routine must be called before using either the CHRIN or the GETIN KERNAL routines for data input. If input from the keyboard is desired, and no other input channels are opened, then the calls to this routine, and to the OPEN routine, are not needed.

When this routine is used with a device on the serial bus, this routine automatically sends the talk address (and the secondary address if one was specified by the OPEN routine) over the bus.

To use this routine:

0) OPEN the logical file (if necessary; see description above).
1) Load the .X register with number of the logical file to be used.
2) Call this routine (using a JSR command).

Possible errors are:

#3: File not open
#5: Device not present
#6: File not an input file

EXAMPLE:

; PREPARE FOR INPUT FROM LOGICAL FILE 2

```
1) LDX #2
2) JSR CHKIN
```

Function name: CHKOUT

Purpose: Open a channel for output
Call address: $FFC9
Communication registers: .X
Preparatory routines: (OPEN)
Error returns: 3,5,7
Stack requirements: None
Registers Affected: .A, .X

Description: Any logical file number which has been created by the KERNAL routine OPEN can be defined as an output channel. Of course, the device you intend opening a channel to must be an output device. Otherwise, an error will occur, and the routine will be aborted.

This routine must be called before any data is sent to any output device unless you want to use the VIC screen as your output device. If screen output is desired, and there are no other output channels already defined, then the calls to this routine, and to the OPEN routine are not needed.

When used to open a channel to a device on the serial bus, this routine will automatically send the LISTEN address specified by the OPEN routine (and a secondary address if there was one).

How to use: Remember: this routine is NOT NEEDED to send data to the screen. 0) Use the KERNAL OPEN routine to specify a logical file number, a LISTEN address, and a secondary address (if needed).

1) Load the .X register with the logical file number used in the open statement.

2) Call this routine (by using the JSR instruction).

```
;DEFINE LOGICAL FILE 3 AS AN OUTPUT CHANNEL
1)  LDX #3
2)  JSR CHKOUT
```

Possible error returns:

3: File not open

5: Device not present

7: Not an output file

B-4. Function name: CHRIN

Purpose: Get a character from the input channel
Call address: $FFCF
Communication registers: .A
Preparatory routines: (OPEN, CHKIN)
Error returns: See READST
Stack requirements: None
Registers affected: .A, .X

Description: This routine will get a byte of data from the channel already set up as the input channel by the KERNAL routine CHKIN. If the CHKIN has not been used to define another input channel, data is expected from the keyboard. The data byte is returned in the accumulator. The channel remains open after the call.

Input from the keyboard is handled in a special way. First, the cursor is turned on, and will blink until a carriage return is typed on the keyboard. All characters on the line (up to 88 characters) will be stored in the BASIC input buffer. Then the characters can be retrieved one at a time by calling this routine once for each character. When the carriage return is retrieved, the entire line has been processed. The next time this routine is called, the whole process begins again, i.e., by flashing the cursor.

How to use:

FROM THE KEYBOARD

1) Call this routine (using the JSR instruction).
2) Retrieve a byte of data by calling this routine.
3) Store the data byte.
4) Check if it is the last data byte (is it a CR ?). If not, go to step 2.

EXAMPLE:

```
        LDX $#00        ;Store 00 in the .X register
1)      JSR CHRIN
2)  RD  JSR CHRIN
        STA DATA,X      ;store the Xth data byte in the Xth
        INX             ;location in the data area.
3)      CMP #CR         ;Is it a carriage return?
4)      BNE RD          ;no, get another data byte
```

EXAMPLE:

```
JSR CHRIN
STA DATA
```

FROM OTHER DEVICES

0) Use the KERNAL OPEN and CHKIN routines.
1) Call this routine (using a JSR instruction).
2) Store the data.

EXAMPLE:

```
JSR CHRIN
STA DATA
```

Function name: CHROUT

Purpose: Output a character
Call address: $FFD2
Communication registers: .A
Preparatory routines: (CHKOUT, OPEN)
Error returns: See READST
Stack requirements: None
Registers affected: None

Description: This routine will output a character to an already open channel. Use the KERNAL OPEN and CHKOUT routines to set up the output channel before calling this routine. If this call is omitted, data will be sent to the default output device (number 3, on the screen). The data byte to be output is loaded into the accumulator, and this routine is called. The data is then sent to the specified output device. The channel is left after the call.

NOTE: Care must be taken when using this routine to send data to a serial divice since data will be sent to all open output channels on the bus. Unless this is desired, all open output channels on the serial bus other than the actually intended destination channel must be closed by a call to the KERNAL close channel routine.

How to use:

0) Use the CHKOUT KERNAL routine if needed (see description above).
1) Load the data to be output into the accumulator.
2) Call this routine.

Function name: CLOSE

Purpose: Close a logical file
Call address: $FFC3
Communication registers: .A
Preparatory routines: None
Error returns: None
Stack requirements: None
Registers affected: .A, .X

Description: This routine is used to close a logical file after all I/O operations have been completed on that file. This routine is called after the accumulator is loaded with the logical file number to be closed (the same number used when the file was opened using the OPEN routine).

How to use:

1) Load the accumulator with the number of the logical file to be closed.
2) Call this routine using the JSR instruction.

EXAMPLE:

```
;CLOSE 15
LDA #15
JSR CLOSE
```

B-10. Function name: GETIN

Purpose: Get a character from the keyboard buffer
Call address: $FFE4
Communication registers: .A
Preparatory routines: None
Error returns: None
Stack requirements: None
Registers affected: .A, .X

Description: This subroutine removes one character from the keyboard queue and returns it as an ASCII value in the accumulator. If the queue is empty, the value returned in the accumulator will be zero. Characters are put into the queue automatically by an interrupt driven keyboard scan routine which calls the SCNKEY routine. The keyboard buffer can hold up to ten characters. After the buffer is filled, additional characters are ignored until at least one character has been removed from the queue.

How to use:

1) Call this routine using a JSR instruction
2) Check for a zero in the accumulator (empty buffer)
3) Process the data

EXAMPLE:

```
;WAIT FOR A CHARACTER
WAIT JSR GETIN
     CMP #0
     BEQ WAIT
```

B-16. Function name: OPEN

Purpose: Open a logical file
Call address: $FFC0
Communication registers: None
Preparatory routines: SETLFS, SETNAM
Error returns: 1,2,4,5,6
Stack requirements: None
Registers affected: .A, .X, .Y

Description: This routine is used to open a logical file. Once the logical file is set up, it can be used for input/output operations. Most of the I/O KERNAL routines call on this routine to create the logical files to operate on. No arguments need to be set up to use this routine, but both the SETLFS and SETNAM KERNAL routines must be called before using this routine.

How to use:

0) Use the SETLFS routine.
1) Use the SETNAM routine.
2) Call this routine.

EXAMPLE:

This is an implementation of the BASIC statement: OPEN 15,8,15,"I/O"

```
LDA #NAME2-NAME     ;LENGTH OF FILE NAME FOR
                    SETLFS
LDY #>NAME
JSR SETNAM
        LDA #15
        LDX #8
        LDY #15
        JSR SETLFS
        JSR OPEN
NAME    .BYT 'I/O'
NAME2
```

RECEIVER/TRANSMITTER BUFFER BASE LOCATION POINTERS

$00F7-RIBUF A two byte pointer to the Receiver Buffer base location.

$00F9-ROBUF A two byte pointer to the Transmitter Buffer base location.

The two locations above are set up by the KERNAL OPEN routine, each pointing to a different 256 byte buffer. They are de-allocated by writing a zero into the high order bytes, ($00F8 and $00FA), which is done by the KERNAL CLOSE entry, They may also be allocated/de-allocated by the machine language programmer for his/her own purposes, removing/creating only the buffer(s)

required. Both the OPEN and CLOSE routines will not notice that their jobs might have been done already. When using a machine language program that allocates these buffers, care must be taken to make sure that the top of memory pointers stay correct, especially if BASIC programs are expected to run at the same time.

ZERO-PAGE MEMORY LOCATIONS AND USAGE FOR RS-232 SYSTEM INTERFACE

$00A7-INBIT Receiver input bit temp storage.
$00A8-BITCI Receiver bit count in.
$00A9-RINONE Receiver flag Start bit check.
$00AA-RIDATA Receiver byte buffer/assembly location.
$00AB-RIPRTY Receiver parity bit storage.
$00B4-BITTS Transmitter bit count out.
$00B5-NXTBIT Transmitter next bit to be sent
$00B6-RODATA Transmitter byte buffer/disassembly location.

All the above zero page locations are used locally and are only given as a guide to understand the associated routines. These cannot be used directly by the BASIC or KERNAL Level programmer to do RS-232 type things. The system RS-232 routines must be used.

NONZERO-PAGE MEMORY LOCATIONS AND USAGE FOR RS-232 SYSTEM INTERFACE

General RS-232 storage:

$0293-M51CTR Pseudo 6551 control register (see Figure 1).
$0294-M51CDR Pseudo 6551 command register (see Figure2).
$0295-M51AJB Two bytes following the control and command registers in the file name field. (For future use.)
$0297-RSSTAT The RS-232 status register (see Figure 3).
$0298-BITNUM The number of bits to be sent/received.
$0299-BAUDOF Two bytes that are equal to the time of one bit cell. (Based on system clock/baud rate.)
$029B-RIDBE The byte index to the end of the receiver FIFO buffer.
$029C-RIDBS The byte index to the start of the receiver FIFO buffer.
$029D-RODBS The byte index to the start of the transmitter FIFO buffer.
$029E-RODBE The byte index to the end of the transmitter FIFO buffer.

commodore
COMPUTER