Serial I/O in Power C

An RS232 function package for C programmers

by W. Mat Waites

One of the most exciting areas of use for the C64 is cheap telecommunications. The ability to communicate with other machines via modem or a hardwired connection adds greatly to the power and value of any computer. The C64 has benefited more than other computers from telecommunications because of Commodore's supplying modems that are very affordable.

Developing telecommunications programs is interesting and fun, but the choices of language for that development have been very few in the past.

Interpreted BASIC is too slow even for 300 baud communication. Compiled BASIC is much faster, but generally utilizes the BASIC interpreter's garbage collection routines for string storage maintenance. The result of this is that the system locks up every few minutes while the string space is being recovered. This will drive you insane after a while.

Assembly language has been the most viable choice for writing programs that would be limited in speed only by the Kernal and the hardware. These assembly programs are very long and difficult to modify, however. The lack of a standardized parameter passing convention and a linker makes it difficult to write functions in assembler that are reusable and sharable with other software developers.

C has come to the rescue with a language that is higher level than assembly, but without the run-time overhead of BASIC and other interpreted languages.

Commodore 64 serial I/O

The C64 actually has a very sophisticated serial I/O system for a microcomputer. It is interrupt driven, which means that incoming characters are taken in by the Kernal even if your program is not quite ready for them. Even such popular systems as MS-DOS do not have this feature. MS-DOS terminal programs must supply their own interrupt-driven code to create this kind of functionality. The Kernal takes care of all of the low level details of accepting characters and sending them out.

Most computers have specialized chips to perform the act of sending out and receiving individual characters. The generic name for this kind of device is Universal Asynchronous Receiver/Transmitter (UART). The C64 emulates the activities of a UART in software. The positive side of this is that software is more flexible than hardware. The C64's serial I/O is at least as configurable as a UART and is more configurable than some. The negative side is that software, especially 1 MHz 6502 software, is slow. The C64 Kernal routines are barely able to keep up at 1200 baud; 2400 baud is not reliable at all.

The two big kludges

There is a problem with the Kernal-supplied serial routines. The timing values supplied for 1200 baud are not exactly correct! The 'width' of the bits coming in to the port don't match up with what the Kernal expects. By supplying new and improved timing values, we can tune the routines to expect the correct bit widths.

This 'bit width fix' introduces another problem, though. With the best possible values in place for receiving characters, there is a problem with transmitting characters. The 'stop bit' (the final bit in a serial character transmission) is too 'narrow'. That is, it doesn't last long enough for the machine at the other end to recognize it reliably. This is not really a problem with simple terminal emulation because no one types fast enough to cause multiple characters to be output one immediately after another.

The short stop bit does cause problems with file transfers. When a block of data is sent, the characters follow one another in rapid succession. The receiver is sometimes still waiting for the end of the stop bit when the next character arrives. This kind of synchronization problem is called a framing error.

To break down this final barrier to reliable 1200-baud communication, a second kludge is introduced. A delay loop is used to wait for each character to be clocked out before another is added to the output buffer. This allows the receiving computer to recover from one character before the next arrives.

Problems with C

In applying C to telecommunications, the first hurdle to overcome is interfacing to the Kernal for several functions. Most

obviously, the serial I/O must be accessed from C. Methods for doing this are not documented in either the Abacus Super-C or Pro-Line/Spinnaker Power C manuals (at least none that I have seen). Other functions that must be implemented include: getting a keystroke without 'hanging', producing a cursor on the screen, doing cursor movement, and providing timing functions for communications protocols.

This article introduces a terminal program written in C and provides the details of the implementation of serial I/O in Power C. *(Note: due to space limitations, only the serial and Xmodem routines themselves are included in the C source listings accompanying this article. The full source for Mat's terminal program, and the program itself, will be included on the* ***Transactor*** *disk for this issue. -Ed.)*

'Packages' of functions

Power C provides an excellent linking facility that allows the programmer to divide his application into as many compilation units as desired. Software development can be made more efficient by writing subsystems that are independent of each other and placing them in separate files. In this way, several different applications may call the same 'package' of functions.

The reusablility of software is very important if you ever intend to develop a software system of any size. You simply cannot start from scratch at the beginning of every project and expect to do large projects.

The package discussed here contains all of the functions and data structures necessary for serial I/O in Power C. It allows you to open the serial device, set the port parameters, write to the serial device, read from the device, and close the device.

The data structures include the input and output queues and the current state of the port.

Opening and Closing the Device

Openserial() opens the C64 'file' for serial communications. Notice that the BASIC-style open() call is used so that a secondary address may be supplied if desired. RS is the symbol for the stream number used for the serial port, 6. With the BASIC-style open, the stream numbers 5 through 9 should be used to avoid conflict with the automatic stream-number allocation done by the higher-level I/O functions.

The closeserial() function simply does a BASIC-style close on the RS stream.

Moving the buffers

The other activity carried out by the openserial() function is moving the buffer pointers to point to the buffers that have been declared for the serial port queues. These are named *inbuf* and *outbuf*. The Kernal allocates the buffers initially at the top of BASIC memory space, but this falls in the middle of Power C space. We simply move the pointers to point to space that we have allocated for this purpose; the rest of memory can then be used without fear of overwriting the input and output queues. This gives the added benefit of allowing the easy examination of the queues without reading characters from them.

Setting the port parameters

After the port is opened, the baud rate and other parameters must be set. We could have specified the baud rate at open time, but we want to be able to change the baud rate at any time so we must work at a lower level.

The setserial() function allows the caller to set the baud rate, the number of bits per character, the number of stop bits, and the parity. This function may be called at any time to change the parameters. The three baud rates implemented are 300, 450, and 1200. Many 300 baud modems will function reliably at 450 baud, and many BBSs support this speed.

Kludge #1 is included in the timing values supplied in this function. The 1200 baud values seem to work well, but they may be tuned for the best performance with your set-up. The 61 may be varied up or down by about 4 or 5.

The stopbits may be set to either 1 or 2. The bits per character may be set to anything from 5 to 8. The parity is set with a bitmap value corresponding to the 3 bits described in the *Commodore 64 Programmers Manual* for selecting parity.

Table of parity values

0 - disabled
1 - odd parity
3 - even parity
5 - mark parity
7 - space parity

Reading and writing

The getserial() function is called to get a character from the serial port. If no characters are available, a -1 is returned. Notice that if the Power C getc() function were called here, the function would not return until a character had come in the serial port. If you are writing a terminal program or BBS, you do not want to 'hang' waiting on characters. You simply want to get it if it is there, or return if it is not.

The putserial() function is called to output a character to the serial port. This function implements kludge #2. There is a delay loop that was shortened until framing errors began to occur. After the loop it simply calls the Power C putc() function to output the character.

Other functions in the package

Functions are also provided in this package for some other DOS-related activities. Functions are provided for accessing the

keyboard without hanging, for checking to see if the'logo' key is pressed (I use this for an attention key), to wait for a given number of seconds, and to read the disk error channel.

Notice that Kernal calls must be made to achieve some of this functionality, but with these functions making the calls for you, you don't need to directly call the Kernal in applications.

Using the package

To use this kind of a package you simply compile it as you would any other function in Power C.

This will produce a file - "dos.o". When you link your application, simply link in "dos.o" and you have serial I/O. Note that you will only compile it once and then link it in whenever you need it. This is a great advantage over BASIC compilers that force you to recompile your entire program every time you make a change.

The terminal program

Included on the disk is a simple terminal program that calls this serial I/O package. It implements a sprite cursor and Xmodem file transfers.

The Xmodem routines are very portable. The Commodore specific I/O functions are separated out and should make it very easy to move the Xmodem part to another operating system.

Ideas for future development

With the serial I/O stabilized, it shouldn't be too difficult to add other protocols: Xmodem CRC, Xmodem batch, Kermit, Punter, and so on. The most difficult thing about implementing some of these protocols is finding definitive documentation. Implementing a BBS is also a possibility.

The Commodore 64 still has a lot of life left in the area of software development. Hopefully, this article will help spur interest in C programming on the 64. Drop me a note if you have any questions, or if you write any interesting applications with the serial package.

I can be contacted on Usenet (!gatech!emcard!mat) or by mail at this address:

W. Mat Waites
1264 Brandl Drive
Marietta, Georgia 30060

```
/* dos.c - operating system stuff:
  disk support
  serial i/o
  kb i/o
  timers */
/* W Mat Waites - Sept 1988 */

#include <stdio.h>

/* 5-9 may be used with "basic" open */
#define KB 5
#define RS 6

/* kludge for reliable 1200 baud */
#define KLUDGE 40

/* kernel routines */
#define CHKIN  0xffc6
#define GETIN  0xffe4
#define TKSA   0xff96
#define ACPTR  0xffa5
#define TALK   0xffb4
#define UNTLK  0xffab

/* input and output serial buffers */
static char inbuf[256], otbuf[256];

/* serial interface functions */

/* openserial() - open serial port */
openserial()
{
  short *ribuf = 0x00f7;
  short *robuf = 0x00f9;

  /* open serial port */
  open(RS, 2, 0, "");

  /* move pointers to buffers */
  *ribuf = inbuf;
  *robuf = otbuf;
}

/* closeserial() - close serial port */
closeserial()
{
  close(RS);
}

/* 300, 450, 1200 implemented */

static short hibyte[3] = { 6,  4,  1};
static short lobyte[3] = {68, 12, 61};

/* setserial() - set serial port */
setserial(bd, bpc, sb, par)
int bd, bpc, sb, par;
{
  short *m51ajb = 0x0295;
  short *baudof = 0x0299;
  char  *m51ctr = 0x0293;
  char  *m51cdr = 0x0294;
  char  *bitnum = 0x0298;
  unsigned indx;

  switch(bd)
    {
    case 300:
      indx = 0;
      break;
    case 450:
      indx = 1;
      break;
    case 1200:
      indx = 2;
      break;
    default:  /* default to 300 baud */
      indx = 0;
      break;
    }

  /* set baud rate */
  *m51ajb = 256 * hibyte[indx] +
    lobyte[indx];
  *baudof = (*m51ajb)*2 + 200;

  /* stopbits */
  if(sb < 1 || sb > 2)
    {
    sb = 1;
    }
  sb--;

  /* bits per char */
  if(bpc < 5 || bpc > 8)
    {
    bpc = 8;
    }
  *bitnum = (char)(bpc + 1);
  bpc = 8 - bpc;

  /* parity */
  if(par < 0 || par > 7)
    {
    par = 0;
    }

  /* put bpc, sb, and par in regs */
  *m51ctr = (char)((bpc << 5)
     | (sb << 7));
  *m51cdr = (char)(par << 5);
}

/* getserial() - char from serial port */
getserial()
{
  int ch;
  char *rsstat = 0x0297;

  ch = getonechar(RS);
```

```
  /* check for empty buffer */
  if((*rsstat & 0x08) == 0x08)
    {
    return -1;
    }
  else
    {
    return ch;
    }
}

/* putserial() - char to serial port */
putserial(ch)
char ch;
{
  int i;

  putc(ch, RS);

  /* delay loop for 1200 baud kludge */
  for(i=0; i<KLUDGE; i++)
    {
    }
}


/* keyboard interface functions */

/* openkb() - open keyboard */
openkb()
{
  char *rptflg = 0x028a;

  open(KB, 0, 0, "");
  /* let the keyboard repeat */
  *rptflg = 0x80;
}

/* closekb() - close keyboard */
closekb()
{
  close(KB);
}

/* getkb() - get char from keyboard */
getkb()
{
  return getonechar(KB);
}

/* charsinq() - # available kb chars */
charsinq()
{
  char *ndx = 0x00c6;

  return (int)*ndx;
}

/* chkstop() - check for <C=> key */
chkstop()
{
  char *shflag = 0x028d;

  return(*shflag == 0x02);
}

/* getonechar() - get char from chan */
static getonechar(channel)
int channel;
{
  char ac, xc, yc;

  xc = (char)channel;
  sys(CHKIN, &ac, &xc, &yc);
  sys(GETIN, &ac, &xc, &yc);
  return(int)ac;
}


/* disk i/o functions */

#define SADDR  0x6f

/* diskerr() - read error channel */
char *diskerr(disk)
int disk;
{
  int cc;
  char ac, xc, yc;
  static char msgbuf[41];
  char *mp;
  char *second = 0x00b9;
  char *status = 0x0090;

  /* tell drive to talk */
  ac = (char)disk;
  sys(TALK, &ac, &xc, &yc);

  /* tell it what to talk about */
  ac = (char)SADDR;
  *second = SADDR;
  sys(TKSA, &ac, &xc, &yc);

  /* read in the response */
  mp = msgbuf;
  for(;;)
    {
    /* get byte from bus in acc */
    sys(ACPTR, &ac, &xc, &yc);

    if(ac == '\r')
      {
      break;
      }
    *mp = ac;
    mp++;
    }
  *mp = '\0';

  /* tell drive to shut up */
  sys(UNTLK, &ac, &xc, &yc);

  return(msgbuf);
}


/* timer functions */

unsigned getclock();

/* sleep() - sleep for seconds */
sleep(usecs)
unsigned usecs;
{
  setclock((unsigned)0);

  while(getclock() < usecs)
    {
    }
}

struct clock  /* struct matches CIA */
  {
  char tenths;
  char seconds;
  char minutes;
  char hours;
  };

/* setclock() - set timer clock */
setclock(usecs)
unsigned usecs;
{
  unsigned bsecs;
  struct clock *clock1 = 0xdc08;
  char *c1mode = 0xdc0f;

  *c1mode &= 0x7f;  /* mode is clock */

  if(usecs > 59) usecs = 59;

  /* convert secs to bcd */
  bsecs = usecs%10 _ ((usecs/10)<<4);

  clock1->hours = 0;
  clock1->minutes = 0;
  clock1->seconds = (char)bsecs;
  clock1->tenths = 0;  /* free clock */
}

/* getclock() - get current clock secs */
unsigned getclock()
{
  unsigned usecs;
  char junk;
  struct clock *clock1 = 0xdc08;

  junk = clock1->seconds;
  usecs = (junk & 0x0f) +
          10 * (junk >> 4);

  junk = clock1->tenths; /* free clock */

  return usecs;
}

/* end of file */


/* xmodem.c - xmodem protocol */
/* W Mat Waites - Sept 1988 */

#include <stdio.h>

/* number of retries, timeouts */
#define RETRY 5
#define TOUT 2
#define BTOUT 10

/* protocol characters */
#define SOH 0x01
#define EOT 0x04
#define ACK 0x06
#define NAK 0x15
#define CAN 0x18

#define RECSIZE 128

char *diskerr();

int rec;
int tries;
int timeout;

/* buffer for data in/out */
char buffer[132];

/* sendfile() - send file via xmodem */
sendfile(fname, disk)
char *fname;
int disk;
{
  int st;
  int ch;
  char errbuf[41];
  char locname[21];
  char *status = 0x0090;
  FILE dfile;
```

```
rec = 1;

strcpy(locname, fname);
strcat(locname, ",r");

/* attempt to open file for read */
device(disk);
dfile = fopen(locname);

/* check for disk error */
strcpy(errbuf, diskerr(disk));
st = atoi(errbuf);
if(st >= 20)
  {
  close(dfile);
  showerr(fname, errbuf);
  return(0);
  }

printf("%s opened\n", fname);

/* clear input buffer */
while(getserial() >= 0)
  {
  }

tries = RETRY;

for(;;)
  {
  printf("Synching...\n");
  if(chkstop())
    {
    close(dfile);
    return(0);
    }
  ch = getchtmo(BTOUT);
  if(timeout)
    {
    printf("Timeout\n");
    tries--;
    if(tries > 0)
      {
      continue;
      }
    close(dfile);
    return(0);
    }
  if(ch == NAK)
    {
    break;
    }
  printf("Strange char [%02x]\n", ch);
  }

printf("Synched\n");

/* send the file */
while(fillbuf(dfile, buffer))
  {
  if(chkstop())
    {
    close(dfile);
    return(0);
    }
  if(!txrec(buffer))
    {
    close(dfile);
    return(0);
    }
  }

/* tell 'em we're done */
putserial(EOT);
for(;;)
  {
  ch = getchtmo(TOUT);
  if(timeout)
    {
    putserial(EOT);
    }
  else
    {
    if(ch == ACK)
      {
      printf("sent EOT\n\n");
      break;
      }
    }
  }

close(dfile);
printf("%s transferred\n\n", fname);
return(1);
}

/* recvfile() - recv file via xmodem */
recvfile(fname, disk)
char *fname;
int disk;
{
  int st;
  int ch;
  int i;
  char r1, r2, dt;
  int response;
  char rchk;
  char locname[21];
  char errbuf[41];
  unsigned chksum;
  FILE dfile;

  rec = 1;

  strcpy(locname, fname);
  strcat(locname, ",w");

  /* attempt to open file for write */
  device(disk);
  dfile = fopen(locname);

  /* check for disk error */
  strcpy(errbuf, diskerr(disk));
  st = atoi(errbuf);
  if(st >= 20)
    {
    close(dfile);
    showerr(fname, errbuf);
    return(0);
    }

  printf("%s opened\n", fname);

  /* clear input queue */
  while(getserial() >= 0)
    {
    }

  /* transfer file */
  response = NAK;
  for(;;)
    {
    /* get a record */
    printf("Record %3d ", rec);
    tries = RETRY;
    for(;;)
      {
      if(chkstop())
        {
        close(dfile);
        return(0);
        }
      /* shake hands */
      putserial(response);
      /* get 1st char */
      ch = getchtmo(TOUT);
      if(timeout)
        {
        tries--;
        if(tries > 0)
          {
          continue;  /* try again */
          }
        printf("Can't sync w/sender\n");

        close(dfile);
        return(0);
        }
      if(ch == SOH)  /* beg of data */
        {
        break;
        }
      else if(ch == EOT) /* done */
        {
        printf("got EOT\n\n");
        close(dfile);
        putserial(ACK);
        printf("%s transferred\n\n",
          fname);
        return(1);
        }
      else if(ch == CAN)  /* cancelled */
        {
        close(dfile);
        printf("Transfer cancelled!\n");
        return(0);
        }
      else
        {
        printf("Strange char [%02x]\n", ch);
        gobble();  /* clear any weirdness */
        response = NAK;  /* and try again */
        }
      }

    response = NAK;
    r1 = getchtmo(TOUT);  /* record number */
    if(timeout)
      {
      printf("TMO on recnum\n");
      continue;
      }
    /* get 1's comp record number */
    r2 = getchtmo(TOUT);
    if(timeout)
      {
      printf("TMO on comp recnum\n");
      continue;
      }

    /* get data */
    chksum = 0;
    for(i=0; i<RECSIZE; i++)
      {
      dt = getchtmo(TOUT);
      if(timeout)
        {
        break;
        }
      buffer[i] = dt;
      chksum += dt;
      chksum &= 0xff;
      }
    /* check for data timeout */
    if(timeout)
      {
      printf("TMO on data\n");
      continue;
```

```
    }

  /* get checksum */
  rchk = getchtmo(TOUT);
  if(timeout)
    {
    printf("TMO on checksum\n");
    continue;
    }

  /* compare rec num and 1's comp */
  if((/r1 & 0xff) != (r2 & 0xff))
    {
    printf("Bad recnum's\n");
    continue;
    }

  /* compare checksum and local one */
  if(rchk != chksum)
    {
    printf("Bad checksum\n");
    response = NAK;
    continue;
    }

  if((r1 ==(rec-1) & 0xff)) /* dupe */
    {
    printf("Duplicate record\n");
    response = ACK;
    continue;
    }

  if(r1 != (rec & 0xff))
    {
    printf("Record numbering error\n");
    close(dfile);
    return(0);
    }

  rec++;

  /* write data to file */
  for(i=0; i<RECSIZE; i++)
    {
    putc(buffer[i], dfile);
    }

  printf("OK\n");
  response = ACK;
  }
}

/* showerr() - display disk error */
showerr(fname, errmsg)
char *fname;
char *errmsg;
{
  erase();
  move(11, 5);
  printf("Error accessing %s", fname);
  move(13, 5);
  printf("[%s]", errmsg);
  move(20, 5);
}

/* getchtmo() - get char w/timeout */
getchtmo(timlen)
int timlen;
{
  int serchar;

  timeout = 0;
  setclock((unsigned)0); /* start timer */

  for(;;)
    {

    serchar = getserial();
    if(serchar >= 0)
      {
      return(serchar);
      }

    if(getclock() >= timlen)
      {
      timeout = 1;
      return 0;
      }
    }
}

/* fillbuf() - get buffer of data */
fillbuf(filnum, buf)
int filnum;
char buf[];
{
  int i;
  int echk;
  char *status = 0x0090;

  for(i=0; i<RECSIZE; i++)
    {
    /* get a char from file */
    if((echk=fgetc(filnum)) == EOF)
      {
      break;
      }
    buf[i] = echk;
    }

  if(i == 0) return 0;

  /* set rest of buffer to CTRL-Z */
  for(; i<RECSIZE; i++)
    {
    buf[i] = (char)26;
    }
  return(1);
}

/* txrec() - send rec, get response */
txrec(buf)
char buf[];
{
  int i;
  int ch;
  unsigned chksum;

  tries = RETRY;

  for(;;)
    {
    /* send record */

    printf("Record %3d ", rec);
    putserial(SOH);
    putserial(rec);
    putserial(/rec);
    chksum = 0;
    for(i=0; i<RECSIZE; i++)
      {
      putserial(buf[i]);
      chksum += buf[i];
      chksum &= 0xff;
      }
    putserial(chksum);

    /* get response */
    ch = getchtmo(BTOUT);
    if(timeout)
      {
      tries--;
      if(tries > 0)
        {
        printf("Retrying...\n");
        continue;
        }
      printf("Timeout\n");
      return(0);
      }

    /* analyze response */
    if(ch == CAN)
      {
      printf("Cancelled\n");
      return(0);
      }
    else if(ch == ACK)
      {
      printf("ACKed\n", rec);
      break;
      }
    else
      {
      if(ch == NAK)
        {
        printf("NAKed\n", rec);
        }
      else
        {
        printf("Strange response\n");
        }
      tries--;
      if(tries > 0)
        {
        continue;
        }
      printf("No more retries!\n");
      return(0);
      }
    }
  rec++;
  return(1);
}

/* gobble() - gobble up stray chars */
gobble()
{
  unsigned gotone;

  printf("\ngobbling\n");

  sleep(2);

  for(;;)
    {
    gotone = 0;
    /* clear input queue */
    while(getserial() >= 0)
      {
      gotone = 1;
      }
    if(gotone)
      {
      sleep(1);
      }
    else
      {
      return;
      }
    }
}

/* end of file */
```