CoCo III
Utilities

CoCo III Utilities

CoCo III UTILITIES is a set of sample programs that provides a practical application of the new features of the Color Computer III and serves as a tool to demonstrate and explain these new functions. To fully utilize these routines and maximize their benefits, the user should first obtain a listing of each program (specific lines of code are referenced to demonstrate new techniques in the text) and then review in sequence the following material (some examples or programs build on previously developed conecpts.)

Your disk contains the following programs:

MEMTEST	128k/512k memory test program.
LOADSAVE	Load & save CoCo 3 hi-res screens.
VERSCROL	Vertical Scrolling Demo.
HORSCROL	Horizontal Scrolling Demo.
CHARPOKE	Change Screen and Character attributes.
CC3WORD	A Simple Word Processor.
CC2TOCC3	Convert graphics & text to CoCo 3.
CIRCLES	Palette registers demo.
SPINBALL	Palette registers demo continued.

MEMTEST 128k/512k memory test program.

Along with the blessing of more memory comes the greater possibility that part of it may be bad. It's simply the law of averages and somewhere down the line, the law will catch someone. The MEMTEST routine is a simple 128k/512k

memory test program, written partially in BASIC with a small machine-language routine that will check the 8K block of memory located at $6000 of the CPU's workspace. The BASIC program will be used to print messages, sequentially swap 8K blocks of memory into the slot at $6000 and execute the machine-language routine to check the block.

LOADSAVE Load & save CoCo 3 hi-res screens.

The new high-resolution screens are fantastic! Very detailed pictures, graphs and charts can be drawn and painted with a variety of different colors. The 320 X 192 screen uses 32K bytes of memory, fortunately this memory is not taken from the BASIC program area. What this means is that your program size doesn't have to suffer anymore when using the new high-resolution screens. It also means however, that you can't directly save the screen to tape or disk. A BASIC program must now be used to save the screen a block at a time. The number of blocks to save is determined by the size of the screen. Remember, each block is 8K in length, so a 32K screen will use 4 memory blocks. BASIC always puts it's graphics screen starting at block $30. So, to save a 32K screen, blocks $30, $31, $32 and $33 would all need to be saved. The **LOADSAVE** routine will illustrate how this is done.

Notice the **OPEN** statement in line 200, it opens a file to the screen. This may seem like a strange thing to do, but it is necessary in this case because the routine that handles the **ONBRK** control does not reset the device number to the screen. Most of the time this will not effect anything, but here the error could occur while accessing the disk which would cause the message in line 200 to be printed to the disk buffer instead of to the screen. Other commands that will reset the device number are **CLS** and **POKE &H6F,0**.

VERSCROL Vertical Scrolling Demo.

The most interesting new feature of the Color Computer III is its ability to smooth scroll in both the Vertical and Horizontal directions. Scrolling is not supported by BASIC except through the use of the **POKE** command.

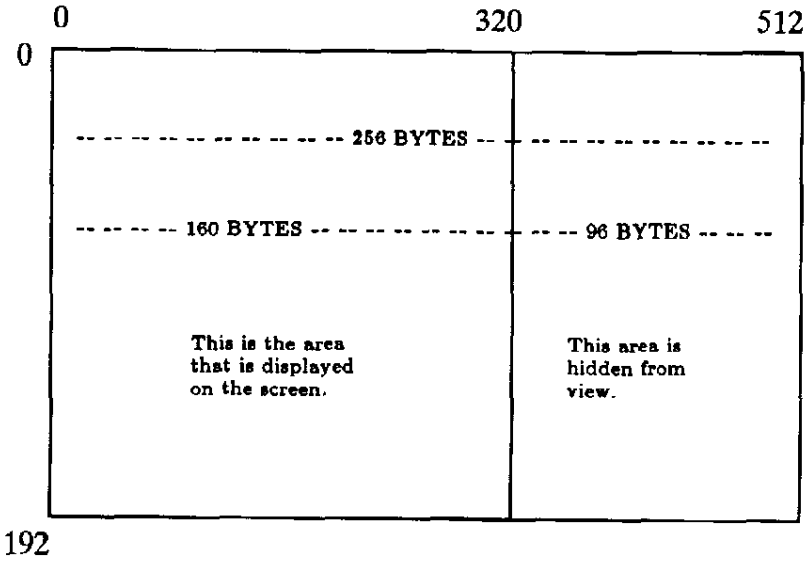
Vertical scrolling is contolled by three registers of the GIME chip, $FF9C, $FF9D and $FF9E. These registers work together to display addresses within the 512K system. In register $FF9C, only 3 bits (5 - 7) are used. Each time these registers are incremented, the display moves by 8 bytes. In order to scroll an entire row, the registers need to be incremented by a value which is equal to the NUMBER OF BYTES PER HORIZONTAL ROW divided by 8. The VERSCROL example will start at BASIC's graphics page ($6000) and scroll the screen according to the position of the joystick. The particular screen being viewed has 160 bytes per horizontal row.

Line 130 is where the registers get incremented, 'S' will equal -1, 0 or 1 depending upon the position of the joystick. This will be multiplied by the number of bytes per horizontal row (160) divided by 8. This value is then converted by the subroutine starting at line 150, into the 3 bytes necessary for storage into registers $FF9C, $FF9D and $FF9E. To make the scroll faster, add the DOUBLE SPEED poke (POKE &HFF9D,0) to line 10.

HORSCROL Horizontal Scrolling Demo.

The horizontal scroll register is located at address $FF9F. Only 7 bits (0 - 6) are used to control the scroll. The eight bit (bit 7) is used to activate the HORIZONTAL VIRTUAL ENABLE (HVEN) mode. Horizontal Virtual Enable uses 48K of memory and is not accessable though

BASIC except with pokes and is required anytime horizontal scrolling needs to do a complete wrap-around. HVEN works by forcing the bytes per horizontal row to 256; the graphics mode selected has no effect on this except to define how much of the 256 horizontal bytes to display. In other words, if a 320 X 192 (160 bytes across) graphics mode is selected while HVEN is turned on, the screen will show the normal 160 X 192 bytes and an area of 96 X 192 bytes will be hidden off the edge of the screen. The following diagram will help clarify this.



The HORSCROL program will set up a horizontal virtual enable screen, clear it with a small machine language routine (BASIC will only clear a 32000 byte screen), **LPOKE** a colored block onto the screen and allow it to scroll according to the position of the joystick.

Line 50 **POKE**s in a small machine language routine that will zero the 8K block of memory located at address $6000 of the CPU's workspace. Line 60 then swaps each 8k block of memory required for the graphics screen into the slot at $6000 and executes the routine to clear it. Notice that at line 160, the Horizontal Offset (HO) is OR'd with $80. This will insure that HVEN will remain set. If for some reason it was desirable to not be in HVEN mode, HO would need to be ANDed with $7F to insure that the HVEN bit was forced off.

CHARPOKE Change Screen and Character attributes.

Changing the attributes of individual text characters (blink, underline, character color and background) plus the number of screen columns are available solutions to older CoCo limitations. The 40 and 80 column screens are located in memory at address $6C000 and is only moved into the CPU's workspace when a character needs to be printed. This is nice because it means that the high resolution text screens do not use any of BASIC's program space. The CHARPOKE routine will **LPOKE** all of the available characters onto the text screen.

The second portion of line 40 **POKE**s in the attribute byte for the character preceding it. You can easily play around with this by changing the value of AT which is set in line 20.

It should be noted that the blink rate of a character that has the blink attribute bit set is controlled by the programmable timer interrupt ($FF94 and $FF95). If both of these bytes are zero'd, the characters will not blink.

CC3WORD A Simple Word Processor.

The program called CC3WORD will give you an example of how the 40 and 80 column text mode commands may be used to create some very powerful programs with almost no effort. CC3WORD is a simple single screen word processor. It allows you to fill the screen with text, save it and print it (press **[BREAK]** to get access to the EXIT, SAVE, LOAD and PRINT options). There are no fancy insert or delete functions, but there is full screen cursor control and typeover.

There are a few interesting things to note about this program. The first is the use of the **ONBRK** command at various lines to change the function of the **[BREAK]** key. Second, is the error trap routine near the end of the program. Third, the SAVE/LOAD routine was designed to work with disk, and can be made to work with cassette by changing the **SAVEM** in line 590 to a **CSAVEM** and the **LOADM** in line 600 to **CLOADM**. Finally, notice the color of the cursor (HINT: it is not the normal underline!).

CC2TOCC3 Convert graphics & text to CoCo 3.

Modufying old programs to work with the new grpahics and text features of the Color Computer III is NOT a very difficult job. In the case of most graphics commands, it is just a matter of adding an 'H' to the beginning of the old command. **PAINT** becomes **HPAINT**, **DRAW** becomes **HDRAW**, **CIRCLE** becomes **HCIRCLE** and so on. In some cases, other changes must also be made. For example, **PRINT@** will not work on the 40 or 80 column screens, **LOCATE** must be used instead. Another example would be when using **HGET** and **HPUT**, instead of dimensioning an array to hold the graphics information, **HBUFF** must now be used to reserve this space.

The **CC2TOCC3** program will aid in converting your old programs. It won't do everything, but it will handle the majority of the work and will flag most of the problem areas. The routine works with DISK ONLY. It reads in a normally save BASIC file and writes out a converted ASCII file. Notice that the command **LPEEK** was included twice in the area for the new secondary functions, this is not a mistake. Due to a bug in BASIC, the new secondary functions skip token 168 and start with 169. The first **LPEEK** is simply a dummy to take this error into account.

CIRCLES Palette registers demo.

One of the dazzling new features of the CoCo III is the ability to display your choice of 64 different colors, 16 at a time on a high resolution screen. the older Color Computers used a technique that fixed specific color definitions to the different screen modes. On the CoCo III, instead of codes defining a color, they point to a location in the input/output space called a palette register that can be individually programmed.

Any palette register may contain any color at anytime. If desired, all palette registers may be set to the same thing. Changing a palette register to a new color will cause all pixels pointing to that palette to instantly change to that color. The BASIC program called **CIRCLES** will draw 4 circles on the screen in different palettes and set them all to the same color. It will then wait for the keys '1', '2', '3' or '4' ot be pressed. Each key will turn on a different circle when pressed and turn it off when released. Notice in line 110 that the last statement is **CMP**. This is the same as the command: **PALETTE CMP**. Also take a peek at lines 2 and 3 for some useful **POKE**s.

SPINBALL Palette registers demo continued.

The **SPINBALL** program is a little more elaborate and involves a fairly long routine. It will build a large ball onto the graphics screen and make it appear to rotate by changing the palette registers through a series of colors. To save space and time, the **DATA** statements from lines 310 to 720 only contain the top left of the ball. The program will take this data and mirror it into a complete ball. The **DATA** statements from lines 240 to 290 contain the ball pattern information and may be changed to create different patterns on the ball. Notice that the last value of each line is an 'F', this determines the palette used for the background area of the ball and should not change. Each of these values is the number of the palette register to use for that area of the ball.