Topic: HOW TO ADD INTERLACED (640 X 450) GRAPHICS TO THE COLOR COMPUTER 3.

Author: GREG BEELEY

Origin: ATLANTA, GA

First BBS: THE ACS BBS (404)-636-2991

Date Sent: JULY 1992

Interlaced graphics is a technique used to double the vertical resolution of a computer's screen. It is accomplished by alternately scanning the even lines and the odd lines. Thus, a full screen is scanned every 1/30 second instead of every 1/60 second. The only disadvantage to this technique is that the screen can have some "flicker" in it, but television uses this technique as well with good results.

Thus, in order to implement this technique on a CoCo 3, one must have a way to make the screen fine-scroll up or down one-half the distance of separation of the normal scan lines. Then, the software needs only to have an interrupt occur every time a screen scan begins and have the interrupt service routine switch the hardware screen start pointer to the screen with the other set of lines (even or odd) and move the screen up or down by 1/2 line. The whole process looks like this:

60hz INTERRUPT!!!

1) Set video start pointer to screen0. (even lines)

2) Set screen at normal position.

3) Clear the interrupt.

4) Return from interrupt.

. . . .1/60 second passes. . . .

60hz INTERRUPT!!!

1) Set video start pointer to screen1. (odd lines)

2) Set screen to 1/2 line lower than usual.

3) Clear the interrupt.

4) Return from interrupt.

. . . .1/60 second passes. . . .

(this process repeats over and over.)

Thus, the way it appears on the screen is as follows: (asterisks are screen pixels)... (lines 0,2,4,etc are the ones that normally would show.)

```
* * * * * * * * * * * * * * * * * * *  (screen line 0) (line 0 of screen0)
* * * * * * * * * * * * * * * * * * *  (screen line 1) (line 0 of screen1)
* * * * * * * * * * * * * * * * * * *  (screen line 2) (line 1 of screen0)
* * * * * * * * * * * * * * * * * * *  (screen line 3) (line 1 of screen1)
* * * * * * * * * * * * * * * * * * *  (screen line 4) (line 2 of screen0)
```

. . .and so forth. . .

The only problem with the CoCo 3 is how to make the screen shift by 1/2 scan line. The GIME chip does not support this function, so some additional-cost hardware must be added. Fortunately, the cost is very low. Here are the components you will need:

NPN transistor (RS catalog #276-2009): 59 cents

10k resistor (RS catalog #271-1335): 39 cents

0.47 uf capacitor (RS catalog #272-1433): 59 cents

10k multiturn trimmer (RS catalog #271-343): $1.49

some wire.

Total cost: about $3.10 to double your resolution!!!

(you may wish to buy a PC board to aid installation. I suggest catalog #276-148. I simply soldered the components together in my version.)

To display the circuit schematic, run "CIRCUIT.BAS" for a HSCREEN1 image of the circuit. Copy down the circuit and directions. Please note that removing your CoCo's cover voids any warranty (who has a warranty now- it has been some time since they stopped selling CoCo's!) By installing this circuit, you assume all risks associated with doing so. Be sure you know what you are doing before you try this. The circuit is simple, though--it is a great one to learn with. Be careful when removing the keyboard--do not tear the plastic connector strip. Handle the keyboard very carefully. And--as always--don't install the circuit while the computer or other equipment is plugged in. We need as many CoCo users as possible. If you or the computer gets fried, there might be a problem with you using your CoCo.

When you solder the wire to the PIA, DO NOT HOLD THE SOLDERING IRON THERE FOR MORE THAN A SECOND OR TWO. A heat sink for soldering is strongly suggested (RS #64-2227 $4.19). You do not want to damage that PIA chip.

The output address is bit 7 of $FF22. To make the screen shift, simply poke (or store) a 1 or 0 in that bit. Note that when you do so, you also invoke some VDG emulation in the GIME. Thus, this circuit can only be used with CoCo 3 graphics (why do you want to use PMODE anyhow?). It will cause the 32-column semigraphics VDG screen to swap between graphics and text 60 times per second. Also, this circuit only works with RGB monitors - the composite and RF outputs are unaffected by the circuit.

This is how the circuit works: capacitor C1 causes a phase shift in the vertical synchronization line to the monitor, which causes the screen to fine-scroll a certain amount. Resistor R2 is set so that the phase shift amount is precisely 1/2 of the normal separation between lines on the screen. Transistor Q1 connects or disconnects R2 and C1 to the ground, enabling or disabling the phase shift in accordance with the state of its base. Resistor R1 connects the PIA output to the transistor, allowing the transistor to be "on" when the PIA output is at 5 volts (logic 1).

Once the circuit is installed, turn the trimmer to about 5k position (turn it any direction until it stops, then turn it back 7 1/2 turns). Then LOADM and EXEC the LINEART.BIN demo. Turn the trimmer until the lines on the screen look "right" and smooth. It may help to turn the contrast all the way down, turn the brightness up until the display is barely visible, and adjust the trimmer until the display looks solid - no individual scan lines can be seen. Even lines which are nearly horizontal should look very smooth.

I would love to see some graphics / text demos show up for the COCO that use this technique (maybe even a 50-line per screen word processor!). Maybe someone could write a graphics drawing program for it. The source for the demo is available and will show how to implement a mapping routine for the interlaced screens. Here is the general technique: (4-color 640 x 450 screen)

1) Verify range of coordinates.

2) LSR the Y coordinate to put screen# in carry.

3) Depending on carry, put screen0 or screen1 in address space.

4) Multiply residue in Y coordinate by 160 and add to that the location of the screen start in memory.

5) Get bits 0 and 1 of X for the position in the byte.

6) LSR X coordinate twice to find actual X value

7) Add X value to Y value to find location of the byte.

8) Use color# and Position# (from step 5) to first AND the byte to clear the pixel and then OR the byte to set the pixel to the right color. See the source to understand how I accomplished this.

One other thing to look at in the source: study the clear screen code segment carefully. It may be nonstandard to use the S stack in this way, but it sure is fast! Compare how fast the screen is cleared in this demo with how fast BASIC clears the HSCREENs. No, I didn't cheat. All the palette registers remained "on" during the clear screen. It just turns out that you can get a very good ratio of bytes cleared per cycle by pushing zeros onto a stack that resides on the graphics page. The ratio is better than 1 byte per 2 cycles! Compare that with the following source code:

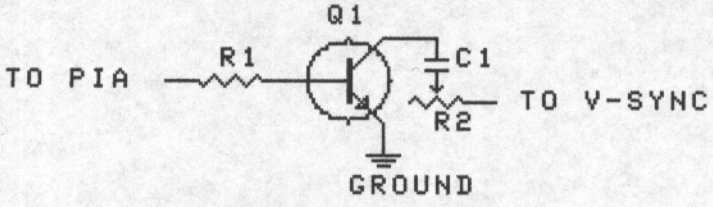
```
100   BEGIN   LDX     SCRNST  Point to screen beginning.

110           LDD     #0      Clear register.

120   LOOP    STD     ,X++    Clear some bytes.

130           CMPX    SCRNEN  Done clearing?

140           BNE     LOOP    Clear more if not.
```

The main loop contains instructions that munch 18 cycles, yet the loop clears only 2 bytes per iteration!!! That's 1 byte per 9 cycles!!! Of course, my method uses more memory... but compare that memory to the amount that the graphics screen consumes! But, memory vs. speed is a trade-off that all programmers have to deal with.

HAVE FUN!!!

INTERLACED GRAPHICS CIRCUIT FOR COCO III

TO PIA: Pin 17 of IC4
TO V-SYNC: Pin 9 of RGB connector
GROUND: Pin 1 or 2 of RGB connector



C1: .47uf Capacitor
Q1: NPN Transistor
R1: 10k fixed resistor
R2: 10k multiturn trimmer resistor