# AFTER THE COLOURS RUSH

This work is a collaboration between Simon Jonassen and Pere Serrat (me). This began in the moment I first contacted Simon to ask some questions related to his 'half-char' shift demo. This genius is also known as the mad guy from the c64 scene that likes to stretch the limits of the Coco II as well ;-)

Both of us have been in search of more colours on the Coco II/Dragon platforms for a LONG time, and various concepts have been tried and tested.

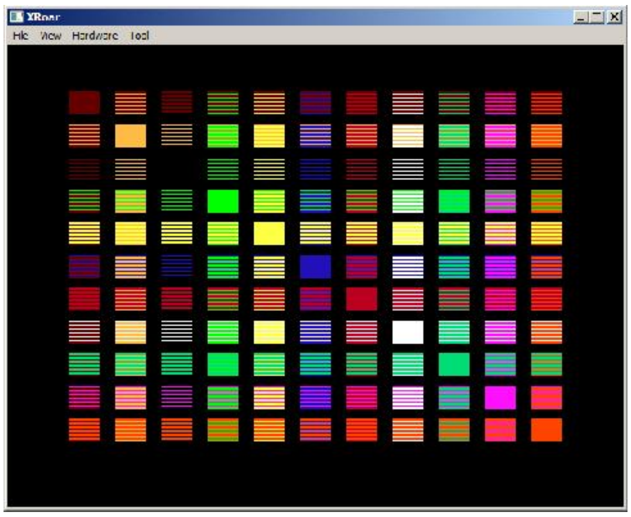
Simon cooked this idea after having taken some inspiration from the current c64 scene where they are doing the same kind of thing.

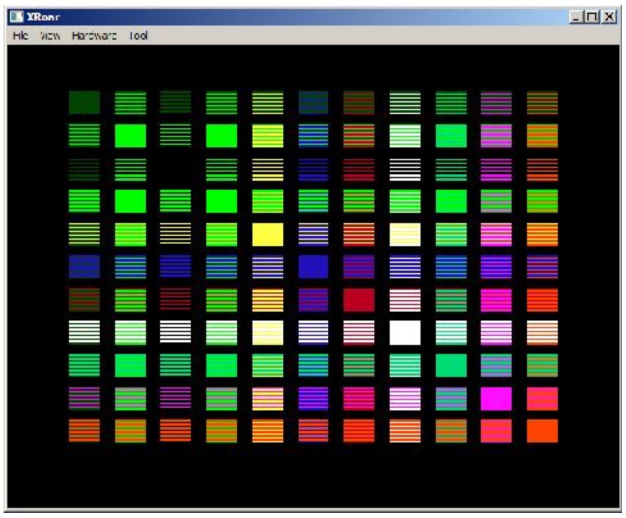
After Simon convinced me that this actually would and could work, the first step we took was based on the semi-graphic mode 24 that allows for the eight standard colours plus black and two spaces, light green and dark green or light orange and dark orange depending on the palette you use.

This means that 11 colours can be mixed one with each other giving 121 possible combinations.

I made the first program to create these 121 combos on the screen, and despite the area for each combination not being very large, you can see some colours blend to give a new colour, yet others don't and simply show a stripped area. I didn't believe it until I saw the result on my Dragon screen!

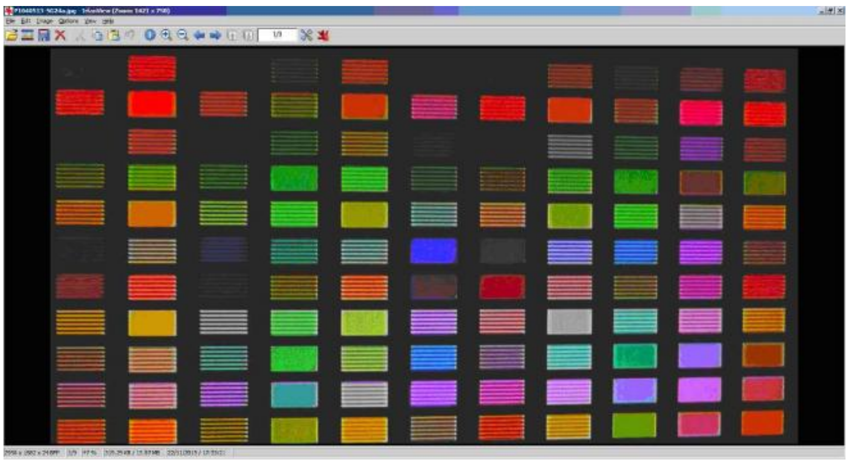
These are the XRoar screenshots for both palettes.





Unfortunately XRoar emulates computer behaviour but not PAL behaviour … (the receiver end), so all stripped combos.

If we run this program on a real computer (CoCo2PAL – Dragon32-64) the result will be like this (bad) photo.



You could ask: Why do some colours blend into a solid colour, while most of the combinations simply show a stripped rectangle? And which ones will successfully blend?

The answer is PAL behaviour.

NTSC creates artifacted colours ‘horizontally’ when a black pixel is next to a white one, creating a light blue or light red if we reverse black and white positions.

Instead, PAL creates ‘artifacts’ vertically due to a correction scheme that is automatically applied.

Technical explanation follows:

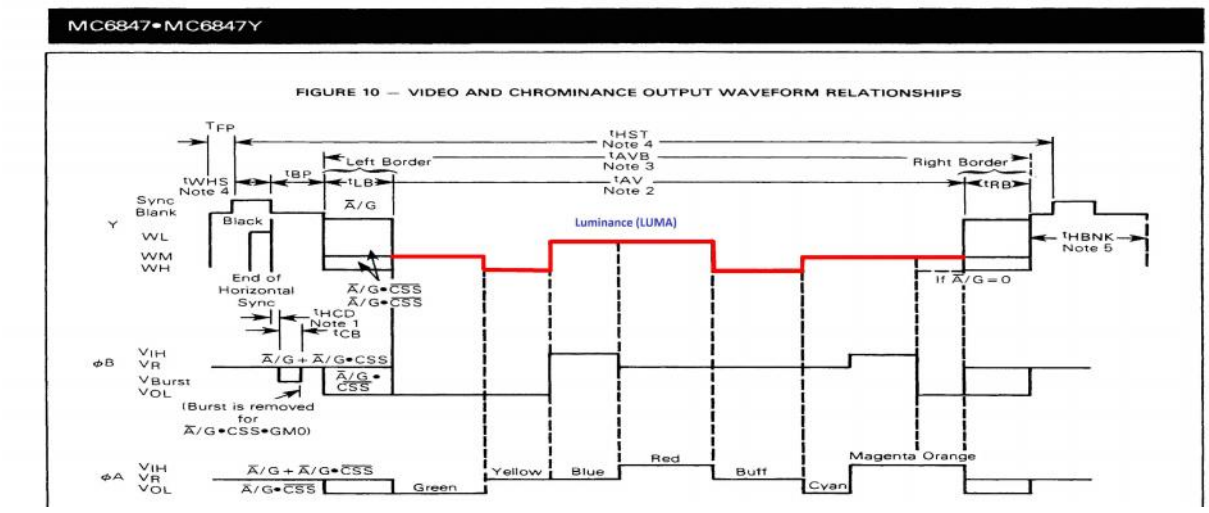
The name ’’Phase Alternating Line’’ refers to the way the chroma (colour) information of the video signal is transmitted, being phase inverted on each line, allowing the automatic correction of the possible phase errors because they cancel each other.

In radio frequency data transmission, phase errors are very common and are due to signal delays at reception or while processing. These phase errors in the transmission of analog video cause an error in the colour tone, adversely affecting the image quality.

Taking advantage of the fact that the colour signal on one line and the next line are usually similar, the receiver automatically compensates colour errors sent to the screen, this gives the mean value of a line and next one, because the possible phase error will be opposite in both lines. That way, instead of showing that error as a tone shift, as it happens in NTSC, we will see a slight colour saturation defect, which is less noticeable to the human eye.

So we are going to take advantage of this automatic correction in the TV, by putting one colour on one line and another one on the next line. The result will be a change of colour for both lines trying to blend, but more like the colour of the first line, so the same colours inverted will show different colour tones. That way, the same two original colours can be used to create two new different combined ones.

To unveil which colours are our ‘friends’ and will create new blended colours, we just have to look at this diagram from the technical datasheet of the VDG MC6847



We can see that there are some colours that share the same luminance (luma) value (red graph).

If we look at the colours of palette 0 (green, yellow, blue and red) we will see that only blue and red have the same luma value, so we could expect two new colours out of them; red over blue and blue over red.

Now let’s have a look at the colours of palette 1 (buff, cyan, magenta and orange). Here we see that cyan, magenta and orange share the same luma value, that’s great because it means that we could get 6 new colours: cyan over magenta, cyan over orange, magenta over orange and their inverses.

Finally we could look for colours that share the same luma value but are in different palette.

In this case we find yellow and buff that will create two new colours and then green with cyan, magenta and orange. This will result in six new colours: green over cyan, green over magenta, green over orange and their inverses.

To summarize this, we can say that in PMODE3 we can expect 6 colours in one palette and 10 in the other, and if we were able to switch of palette every horizontal scanline, then we could get 8 extra colours, so raising the total to 24 colours at the same time on the screen!

As we have seen that SG24 adds three more colours, then three new possibilities arise:
Dark orange / black, light orange / yellow and light orange / white

And in this case we don’t need to switch palette because the eight colours are on hand. Anyway if someone wanted to do it, he could use the other two spaces (light green, dark green) to create even more colours.

If you want to give it a try, here is a VDK file that contains all the needed programs. You just have to run the basic program “TSTSG24.BAS” on a real Dragon / CoCo2 PAL to see these new colours.

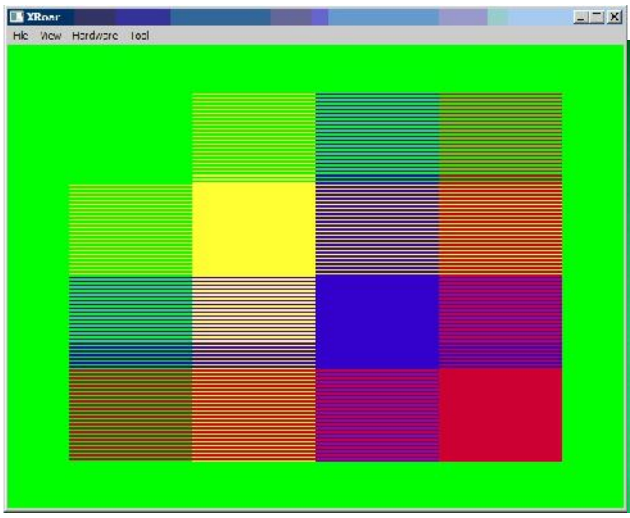
First the colours in palette 1 are shown (using light orange and dark orange). Pressing ‘C’ will show the ones in palette 0 (using light green and dark green).

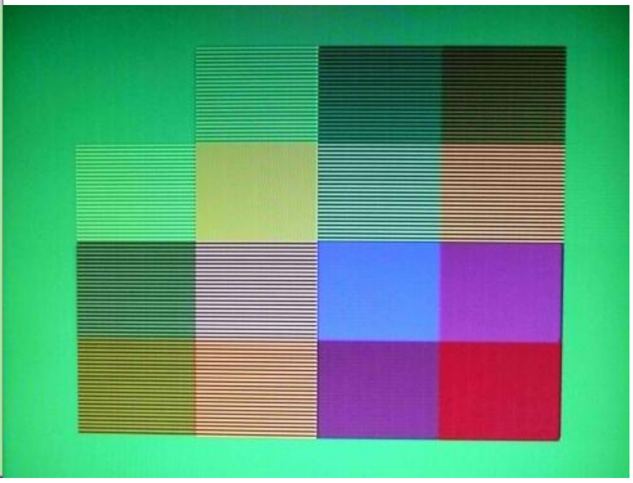
Each new ‘C’ keypress will alternate between them. To quit the program just press Break.

INCLUDE THE FILE “25A - Blending Colors in PAL.VDK”

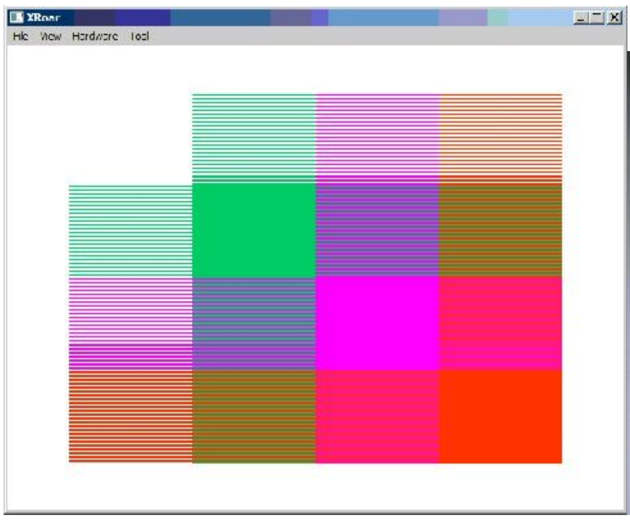
Next step was to apply this technique to PMODE3 to verify the results. The next two images show XRoar screens and a real Dragon using palette 0. You can see the two new combinations on the photo (red-blue combos at the lower right of the screen).

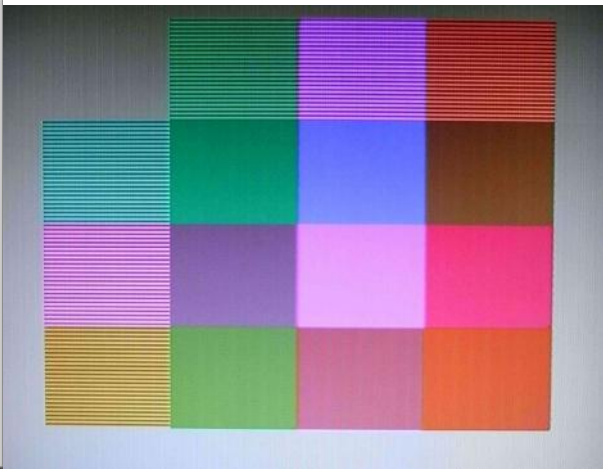
The photos do not capture the exact colours but gives you an idea of the principle. On the Dragon screen the two purples have different colour tones (darker – lighter).



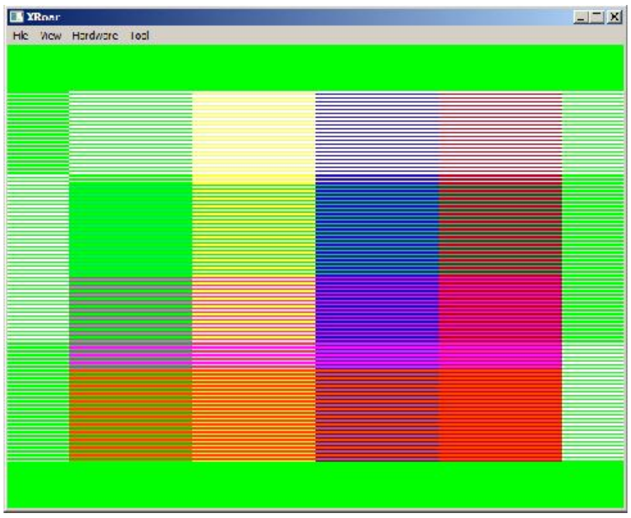


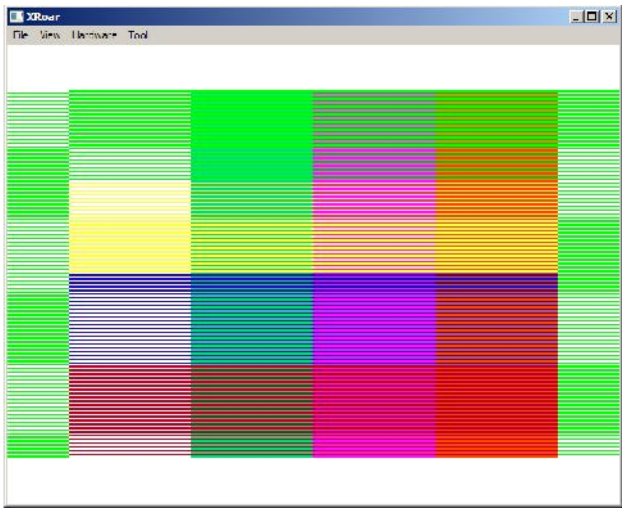
Now the images using palette 1. In both XRoar and on a real Dragon.  The photo you can see at the bottom right of the screen has NINE solid colours!, six of them are the blending result of two standard colours.





And this is a mix of both palettes, the borders show a switching of colour on every line. XRoar images only,
On the left first CSS1 then CSS0 and on the right first CSS0 then CSS1 (symmetric results here)



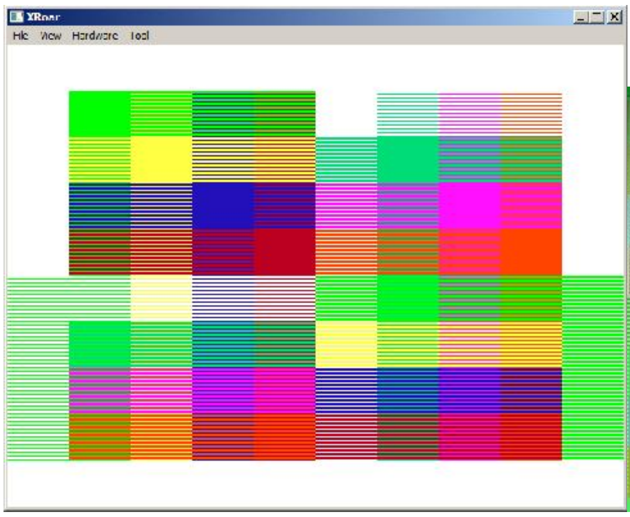


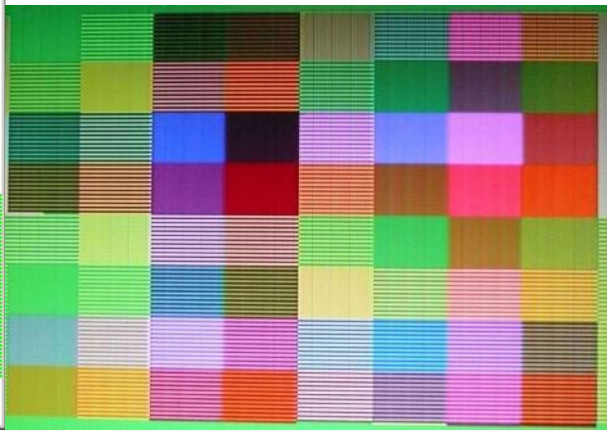
To see these images on your computer, you just have to use the basic programs “TSTBND10.BAS” and “TSTBND01.BAS” respectively, both contained in previous VDK file.

To end the demos you should have to press Reset.

The next challenge was to show these four palette combinations on just one screen.

The result is shown in this XRoar screenshot and D64 photo





Upper left quadrant has combinations of colours of palette 0
Upper right quadrant has combinations of colours of palette 1
Bottom left has combinations of palette 1 and then palette 0
Bottom right has combinations of palette 0 and then palette 1

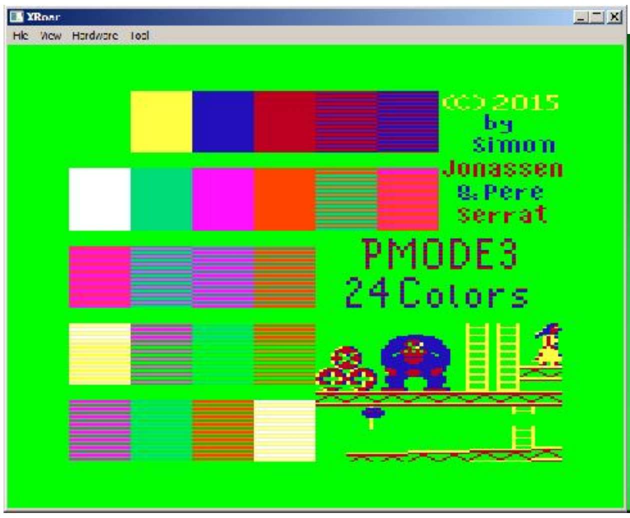
To test this image you can use the basic program “TST4Q.BAS” from the disk file “25B - 4 Quadrants v1.0.VDK”

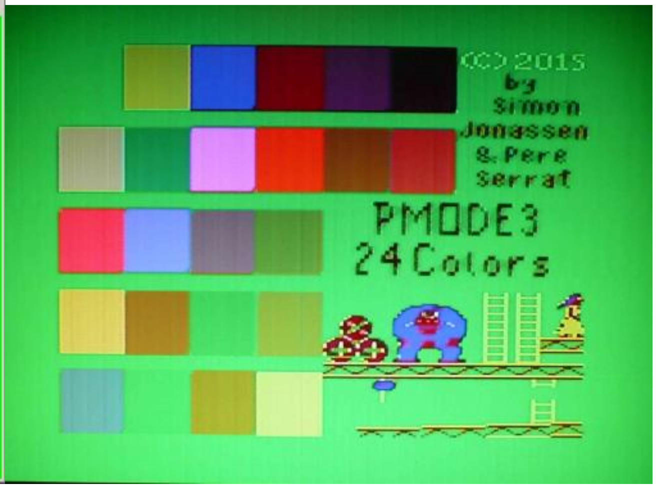
INCLUDE THE FILE “25B- 4 Quadrants v1.0.VDK”

Now, despite the fact we can see the 24 solid colours we are able to obtain, we want to get rid of the combinations that don't blend into a new colour, so that we only have the good ones.

Here are the XRoar screenshot and a photo of the real D64, a very poor photo due to the moiré effect added by the camera, sorry.

The mix of text, Donkey and the patterns to be blended were made by me.
To see that image you have to execute the program "TST24M.BAS" from the disk file named:
"25C - PMode3 with 24 colors x Dragons v1.3.VDK" for Dragon Users, or
"25C - PMode3 with 24 colors x CoCo v1.3.DSK" for CoCo2 PAL users





Clearly the switching of palettes on a scanline basis is only useful for still images, not for moving elements (games).

For still images we have to create a converter to map the 24 colours from the original image to the 24 we can obtain with the methods used in previous images.

But for games we only have to use one palette, and if palette 1 is adequate then we could use up to 10 colours, not that bad, don't you think so?

The problem here with mixing the PMode3 fixed image and text, both palette 0 with the rows of combos that have different palettes is that you have to synchronize your code with the horizontal frequency (each line sets an irq flag) as well as each frame.

Simon and I have spent numerous hours discussing timing and the concepts involved in pulling this off.

The first row of combos is palette 0 only, so no changes need to be done.

Second row is palette 1 and shows six combos, so we have to change to palette 0 exactly after the sixth 'new' colour.
Third row is also palette 1 but shows only four combinations, so the change to palette 0 has to be made exactly in the middle of the line.

Finally fourth and fifth rows have alternated palette every line but just until the centre, then palette 0 must be set for both even and odd lines.

This implies accurately cycle counting when coding, adding the needed dummy opcodes to ensure that changes happen in the right moment.

Furthermore, we decided that the executables (pattern generator and palette switcher) had to work in CoCo2 PAL, in Dragon 32 and in Dragon64 as well.

Every machine behaves differently from the others in some aspects, for instance:

CoCo2Pal and Dragon 32 send 312 HS IRQs, so 50 more than Dragon 64 that just sends 262.
CoCo2Pal has an overall different timing and needs different dummy opcodes to sum up a different value for each palette transition.

Simon added a routine that detects the machine type and code self-modifies some parts of the code to accommodate for the detected machine. The programs will even run on a CoCo2-NTSC machine but nothing special is going to be seen because of the NTSC video output …

INCLUDE THE FILE "25C - PMode3 with 24 colors x Dragon v1.3.VDK"

INCLUDE THE FILE "25C - PMode3 with 24 colors x CoCo v1.3.DSK"

# S·Y·N·C·H·R·O·N·I·C·I·T·Y

When we want a change of palette (CSS) at a certain byte on one line, we must count CPU cycles so that the opcode that modifies the palette hits on the exact cycle that corresponds to the desired screen byte.

We need to know some figures before we put our hands on the calculator …
Here comes a series of data from the VDG MC6847 datasheet. Simon helped me finding and explaining it.

In page 7 of that technical document we can find this information as note 6 in figure7:
One full scanline lasts 227,5 cycles of the chroma clock that runs at 3,5795MHz.

As we know that the MC6809 CPU runs at 0,8948MHz (exactly one fourth of the chroma clock), we can tell that one screen scanline lasts 227,5/4 CPU cycles, that gives roughly 57 cycles.

In figure 11 on page 9 of the same datasheet, we could read that the visible line lasts 128 chroma cycles or 32 CPU cycles.

This gives 25 cycles (bytes) for the ‘non visible’ part of the line which is composed of parts like:

Horizontal sync pulse (/HS), front porch, back porch, left border, right border, colour burst.

The duration of each one of these components can be seen on the table on page 5 of that document.

So it seems that from the HS detection until the beginning of the visible area there are 13 cycles, so the part on the right of the visible area should sum up to 12 cycles that together with the 32 of the line result in the known 57 cycles.

Now we see that in order to change palette at byte ‘one’ we should hit cycle 13+1

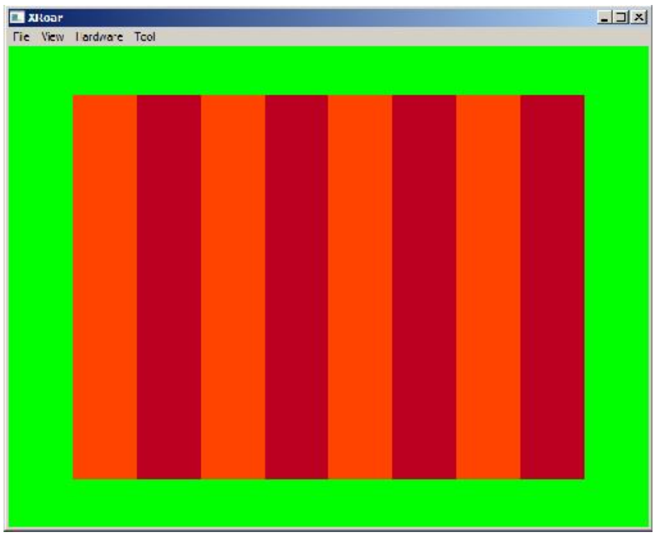
For byte 2 it will be 13+2 and so on until byte 32 that would be 13+32= cycle 45

Let’s imagine we want 8 vertical bars equal width, so 4 bytes each, then we should hit these
cycle numbers: 14, 18, 22, 26, 30, 34, 38 and 42 corresponding to bytes 1,5,9,13,17,21,25,29 on the line
Let’s write a small assembly routine that does exactly that:

Assuming we have loaded previously register D with values $e8e0 for CSS1 and CSS0 and DP=$FF

```
hloop   nop                     ; (02)(02)
        exg     a,a             ; (08)(10)
        sta     <$22            ; (04)(14)      CSS=1
        stb     <$22            ; (04)(18)      CSS=0
        sta     <$22            ; (04)(22)      CSS=1
        stb     <$22            ; (04)(26)      CSS=0
        sta     <$22            ; (04)(30)      CSS=1
        stb     <$22            ; (04)(34)      CSS=0
        sta     <$22            ; (04)(38)      CSS=1
        stb     <$22            ; (04)(42)      CSS=0
        tfr     a,a             ; (06)(48)
        tfr     a,a             ; (06)(54)
        bra     hloop           ; (03)(57)
```

The first number in parenthesis is the number of cycles used by that opcode, the second is the accumulated cycle count along the routine.
You see that every ‘st’ opcode to $ff22 is exactly on the cycle count pre-calculated. This would show that image:



Besides the exact cycle count to hit the correct byte to have the right switching, you can see that we have added ‘dummy’ cycles to sum a total of 57 cycles so that without any synchronization of interrupts the program will be processing all of the lines … even the non visible, of course, if this doesn’t matter as is the case.

The cycle counts here are ONLY for the Dragons. If you wanted to do the same in a CoCo2 PAL, then you would have to reduce that cycle count by 4, so we should hit instead these other cycle numbers: 10, 14, 18, 22, 26, 30, 34 and 38.

This implies that we’d have to adjust the time-waster opcodes before the first change and after the last one.
EXG a,a should be converted into two NOPs (2+2 cycles instead of 8) and the two TFR a,a converted into two EXG a,a (8+8 instead of 6+6) because the loop must sum a total of 57 cycles in any case.

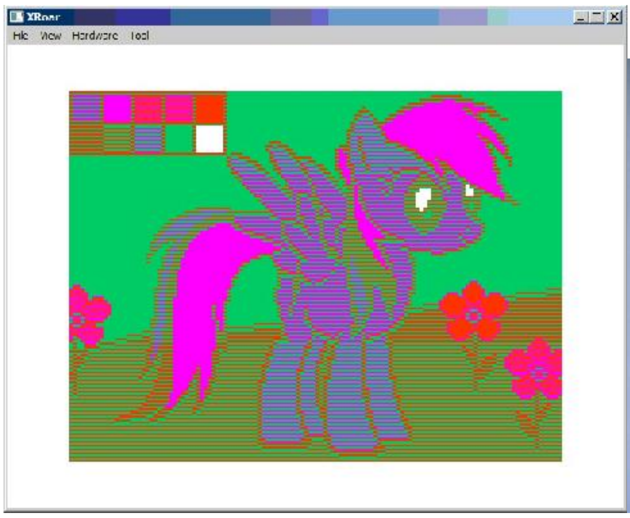
## E X A M P L E S   OF   U S E

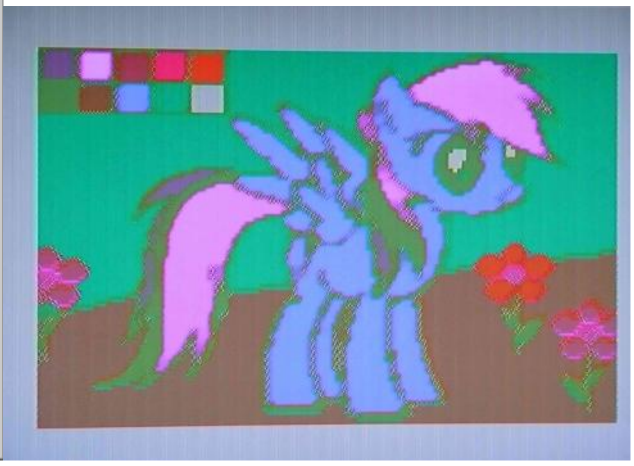
Here are two images, a 'pony' and  a 'testcard', that will allow the user to modify the colour settings on the TV set so that the 10 colours are clearly visible. Both XRoar screenshots and photos of real hardware are shown.

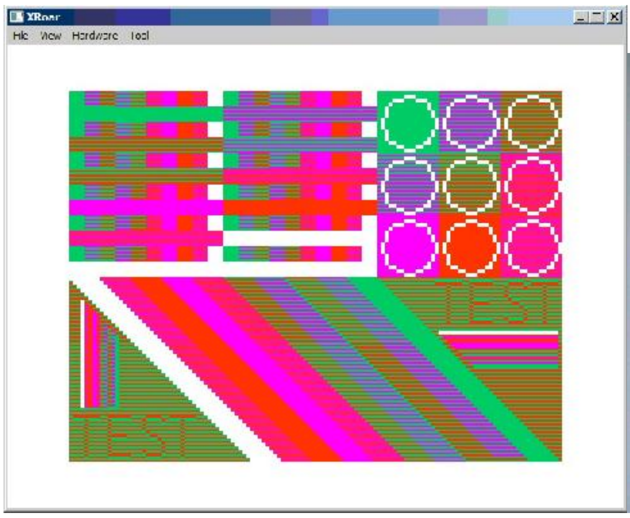
The images were designed by Hugo Dufort in format .PNG and later converted by Simon to CoCo/Dragon images with his own converters.

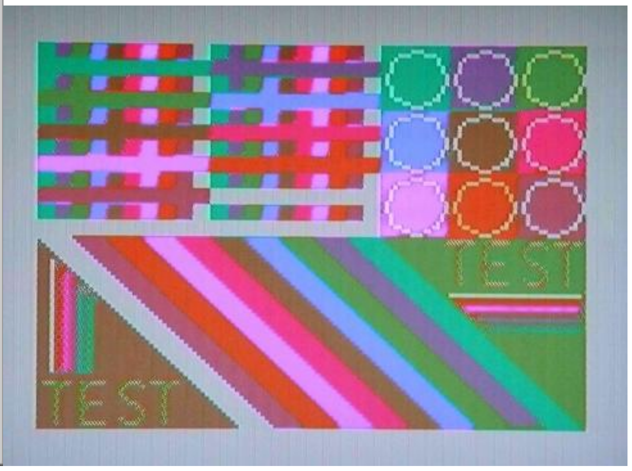
The camera has captured the colours its way, sadly the result is worse than the image shown on the real hardware.

Anyway the pictures give an idea and are a proof of the concept. Using PMODE3 in palette 1 you can get up to ten different solid colours at the same time on the screen.









You can see these images on your computer running the program "SEEIMGS.BAS" included in last VDK.
It is in both versions, Dragon and CoCo disks.

These works have been made just to show Dragon/CoCo users that have a PAL device, that many more colours could be used using colour graphic modes and any semigraphic mode as well.

Hopefully someday some among you will come with new developments exploiting those possibilities that have been for so many years there waiting for someone to use them.

Cheers
Simon Jonassen & Pere Serrat

## NOTES

These are some interesting sites where you could find information about the 'colours' subject

The thread opened by Simon on worldofdragon with the title "28+ colors on coco II /DRAGON (NO FLICKER)"
http://archive.worldofdragon.org/phpBB3/viewtopic.php?f=5&t=5450

Another thread that shows ideas to get more colours, but in this case by mixing two images can be seen at
http://archive.worldofdragon.org/phpBB3/viewtopic.php?f=8&t=4948

Other known projects related to this work that were made previously:
John W. Linville:  44+ colours and 35+ colours techniques.
See this url: http://vdgtricks.blogspot.com.es/

Nickolas Marentes game "Donut Dilemma"
see this url:
http://www.members.optusnet.com.au/nickma/ProjectArchive/donutcc.html