

A World Without NTSC

Bridge the Gap Between NTSC and VGA

NTSC will soon be a thing of the past. So, what will you do in a world without the NTSC? Jeff answers that question and more. Read on to learn how he is using a chip to bridge the gap between the NTSC and VGA formats.

The United States Federal Communications Commission (FCC) has mandated that most broadcasters cease transmitting NTSC in favor of digital television. What will happen in a world without NTSC?

I was raised on NTSC. My uncle Ray was the first in my family to have a color TV. I remember saying, to my uncle's dismay, "I prefer black and white to color; look how awful the picture is." The grainy, fuzzy, rainbow-colored objects were tough to watch. And I'll admit now that this may have been due to early set design and fringe reception. Back then, we were considered fortunate if

we could receive all three major networks. Today's TVs (or should I say those of the recent past) do a great job at receiving broadcast signals. Strong stations give crystal-clear pictures. I don't know the exact numbers, but many viewers have now given up their antennas for cable or dish connections. Their broadcasts are already digital. Their receiver boxes translate the ones and zeros into NTSC output so we can connect our legacy TVs. For those of you still using an antenna for reception, the new digital broadcast transmissions can not be received directly by legacy TVs. They require a converter box to stupefy the new broadcast format down into an NTSC output that can be used by the outdated equipment.

I'm not going to debate the pros and cons of the new digital broadcast format. Instead, I want to point out that this means an end to using inexpensive NTSC TVs and monitors as display devices. All of this may have started back with Don Lancaster's design of the TV typewriter that appeared on the cover of *Radio-Electronics* magazine in September 1973.^[1]

NTSC is a composite video standard used by the first personal computers (TRS-80 and Apple) and video game systems (Coleco and Atari). As higher resolutions were required, the composite video signal was separated into multiple components allowing finer control of the video format. While (S)VGA uses discreet signals, each color is still basically analog. One of the newer standards, the Digital

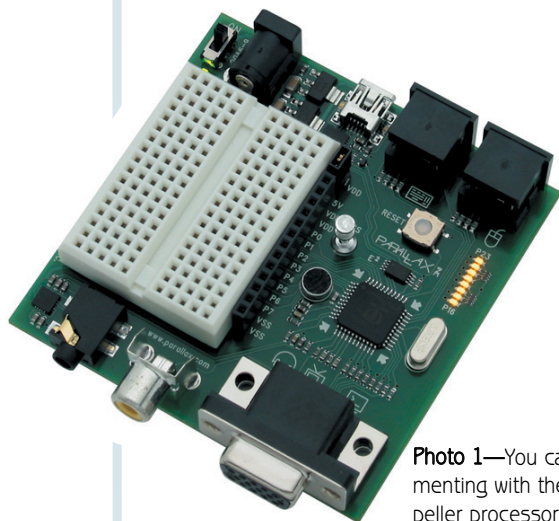


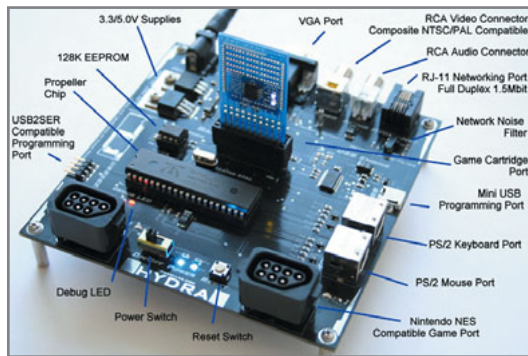
Photo 1—You can start experimenting with the Parallax Propeller processor for \$80 using the Propeller demo board.

Video Interface (DVI) combines DVI-D (digital mode) and DVI-A (VGA in analog mode). Of the advertised interfaces on today's TVs—such as, HDMI, component, VGA, S-Video, RF, and composite—which option do you think will be the first to go on future models?

PROPELLING

I spent most of my early hard-earned pocket money playing Space Invaders and Asteroids at the local hangout. I thought I had since shed my addiction for gaming. Little did I realize the demon had only moved into the shadows. I continually collect products and technologies that I think have potential

Photo 2—The HYDRA game console is based on the Propeller chip. The experimental console comes complete with a P52 mouse, a P52 keyboard, a game controller, a power supply, and cables. It also includes the book *Game Programming for the Propeller-Powered HYDRA* and a CD for \$200.



for future spotlight time in one of my monthly raves. For instance, after having a Parallax Propeller chip sitting on my shelf for a few years—along with Andre LaMothe's *Game Programming*

for the Propeller-Powered HYDRA—I recently started playing with it. I read "GPFTPPH," but it wasn't until I saw the FCC's writing on the wall that I understood how the Propeller could

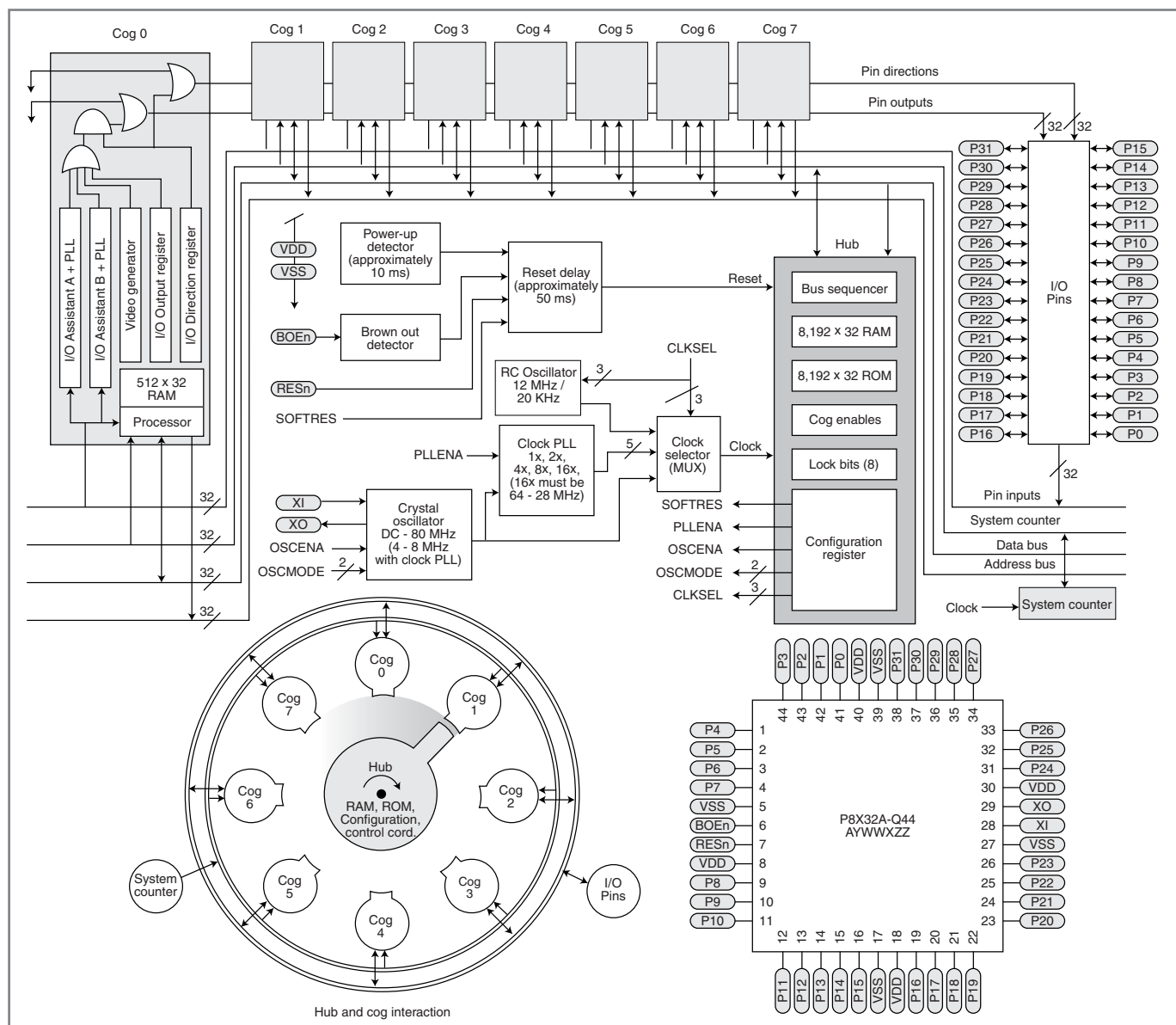


Figure 1—The Propeller block diagram shows the interaction between the hub and eight cogs. Each cog has access to all I/O pins and the hub's RAM and ROM, as well as its own RAM.

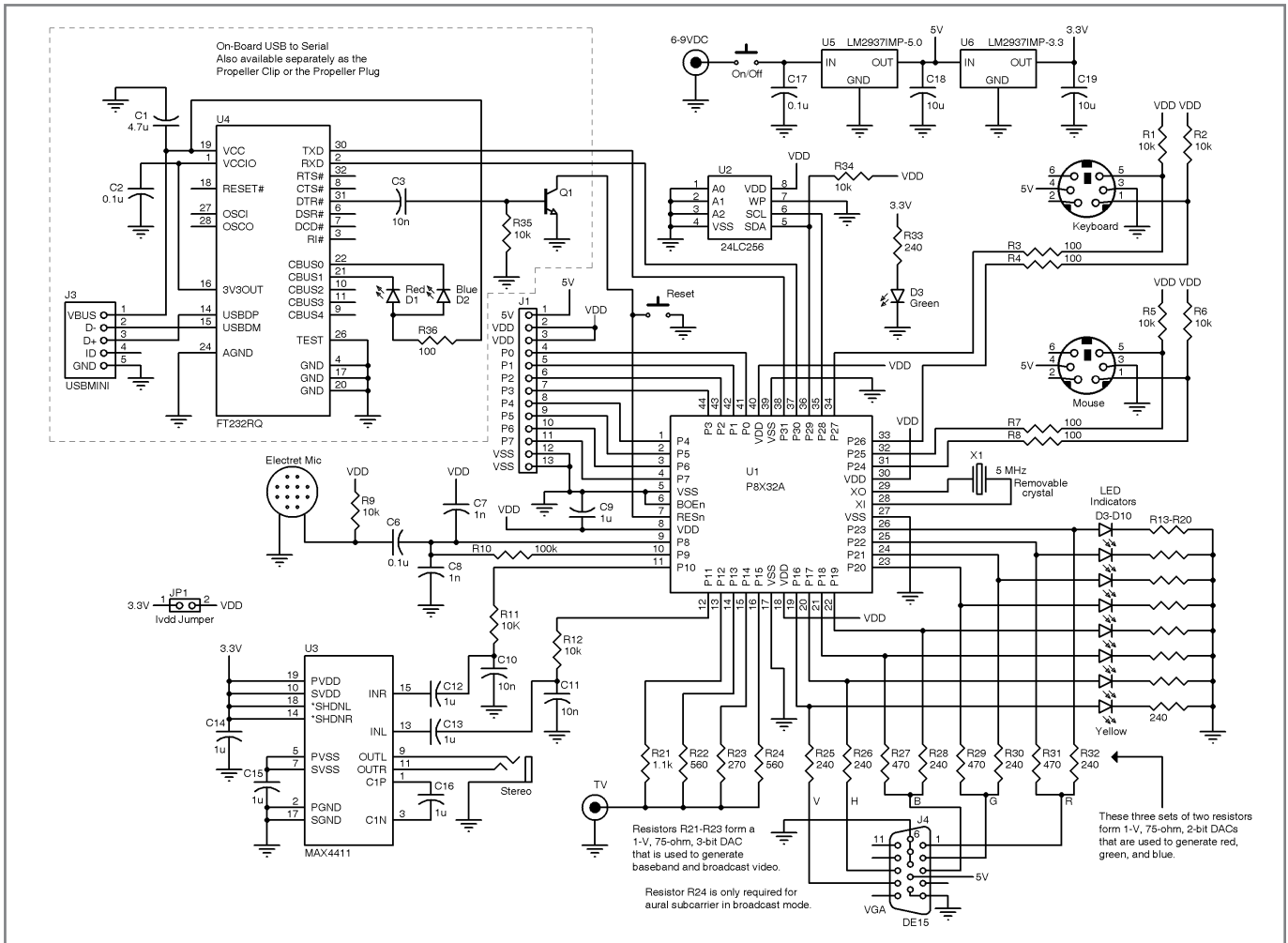


Figure 2—The Propeller demo board schematic shows how simple resistors added to the Propeller’s I/O pins act as a DAC (with a monitor’s 75-Ω input impedance) creating an inexpensive interface for either NTSC or VGA video.

easily bridge the gap between NTSC and VGA. So, I took my Propeller demo board and went to work (see [Photo 1](#)). The HYDRA is shown in [Photo 2](#).

The Propeller chip consists of eight independent processor units called cogs. Each cog has its own 512 double words (32 bits) of RAM. The last 16 bytes of this RAM are special function registers that enable the cog to access all I/O pins, its own counters, and a video generator. The RAM is used to hold code and local VARs. Each cog executes its own code independently, yet all cogs run from the master clock. The Propeller can take oscillator or XTAL input or use an internal RC oscillator to drive the internal clock directly or through a 2x/4x/8x/16x PLL for a maximum clock speed of 80 MHz!

Note that in addition to eight cogs, there is an additional device called the hub (see [Figure 1](#)). Besides handling the

basic reset, brown out, and master clocking, the hub has its own memory, both RAM and ROM, with 8,000 double words each. Hub ROM contains character definitions, math functions, a bootloader, and a Spin interpreter. During reset, the bootloader loads COG0 with some code that checks for communication (enabling you to take control), checks for an external EEPROM, and loads it into the hub’s RAM or shuts down all operations. If an application has been transferred into the hub’s RAM, then the SPIN interpreter is loaded into COG0 and begins to execute the application in the hub’s RAM.

Like all microcontrollers, the Propeller has a number of assembly instructions that make up its vocabulary. You may want to write your application code (or parts of it) in assembly language. However, there are those who detest having to work with assembly code, so the Parallax folks created a

higher-level language called Spin. It removes much of this burden by providing a bunch of useful functions.

While each cog executes on its own, your application directs this operation and will determine exactly how a cog will be used. For instance, if your application requires asynchronous serial communication, you might write a cog application that samples the RX input looking for a start bit, and upon reception uses the system clock to continue sampling the input at the proper data rate. Collected bytes might be put into hub RAM (available to any cog). The hub continuously does a “round robin” on all of the cogs. It controls when a cog has access to the system RAM and keeps cogs from simultaneous access. A cog may have to wait its turn, which is a maximum of once every 16 clock cycles, depending on whose turn it is. If a cog needs to update multiple RAM locations prior to allowing

Bits	Video configuration register (VCFG)			
31	n/a			
30:29	VMode (Enable)			
	0	0	Disable VSU	
	0	1	VGA Mode	
	1	0	NTSC Mode (Broadband on upper pins, baseband on lower pins)	
	1	1	NTSC Mode (Baseband on upper pins, broadband on lower pins)	
28	CMode (colors/shades)			
	0	Two-color mode		
	1	Four-color mode		
27	Chroma 1 (Broadcast)			
	0	Disable chroma (color) on broadband		
	1	Enable chroma (color) driver		
26	Chroma 0 (Baseband)			
	0	Disable chroma (color) on baseband		
	1	Enable chroma (color) driver		
25:23	Aural subcarrier (source)			
	0	0	0	Use COG 0's PLLA
	0	0	1	Use COG 1's PLLA
	0	1	0	Use COG 2's PLLA
	0	1	1	Use COG 3's PLLA
	1	0	0	Use COG 4's PLLA
	1	0	1	Use COG 5's PLLA
	1	1	0	Use COG 6's PLLA
	1	1	1	Use COG 7's PLLA
22:12	N/A			
11:9	VGroup (Port drive)			
	0	0	0	Group 0 (P7:0)
	0	0	1	Group 1 (P15:8)
	0	1	0	Group 2 (P23:16)
	0	1	1	Group 3 (P31:24)
	1	x	x	Reserved
8	n/a			
7:0	VPins (Pin drive)			
	0	1111		Driving lower four pins only (NTSC)
	1111	0000		Driving upper four pins only (NTSC)
	1111	1111		Driving all eight pins (VGA)

Table 1—This 32-bit register defines how the VSU hardware is used. The upper bits define the NTSC/VGA modes, resolution, chroma, and audio carrier source, while the lower bits define which processor pins are used for output.

others to access the data, it can indicate this with a lock-flag. Other cogs should respect this flag and cease access until it is cleared.

There are no interrupts on the Propeller. Think of a cog as an interrupt routine that continuously executes its code, independent of other cogs. Every cog can read and write to every I/O at any time! You can use this to your advantage, but without proper attention, it can cause you headaches. Any pin configured as an output by one cog will force the configuration of the pin to an output even if another cog is trying to use the pin as an input. Any cog outputting a high on a pin will force the pin high even if another cog is outputting a low to the same pin.

The hub's RAM (\$0000-\$7FFF) will hold your application after it is transferred at boot time. The hub's ROM code consists of 256 printable characters and graphics (\$8000-\$BFFF), Log and Anti-log tables that help convert between base-2 and

floating point (\$C000-\$DFFF), a sine table for 0° to 90° with 0.0439° resolution (\$E000-\$EFFF), and the bootloader/Spin interpreter (\$F000-\$FFFF).

VIDEO HARDWARE

Each cog has its own video hardware consisting of two configuration registers and the ability to stream data using Propeller output pins via the video streaming unit (VSU). (Note that while the primary use here is video, don't overlook the audio possibilities.) The digital outputs are meant to interface to an external DAC producing composite NTSC video output. Because a composite monitor presents a 75-Ω load, discrete resistors can be used to implement a DAC (see Figure 2). The VCFG 32-bit register is used to configure the VSU in a number of modes (i.e., Composite Baseband, Composite Broadband) (55.25 MHz = channel 2) or VGA (see Table 1). The VSCL 32-bit register contains two values: the number of CTLA PLL clocks per pixel (PixelClocks), and the number of clocks per frame (FrameClocks) (see Table 2). A CTRA PLL clock is based on your XTAL value, the PLL multiplier, and the CTRA PLL divider. When using NTSC, the active pixel area of a scan line is 52.6 μs. If you want to divide this into 256 pixels, that's approximately 205 ns/pixel (52.6 μs/256 pixels). If the CTRA PLL clock is running at 40 MHz, that's 25 ns (1/40,000,000). The closest you could come to 205 ns would be to use a count of eight CTRL PPL clocks. That would be 200 ns (PixelClocks = \$08). If you were using 2-bit (four-color) mode, you would be storing 16 pixels worth of 2-bit information in each 32-bit double word. This means that the FrameClocks value would need to be 16 pixels × PixelClocks, in this case 8 (FrameClocks=\$080).

With these registers set up, the cog has all of the timing information it needs to automatically output a stream of data via selected output pins. But what about the data that needs to be moved? The data will come from a RAM buffer. The Propeller has 32 KB of RAM for system use. This includes variable storage, program storage, and stack space, so you have only a fraction for video data. Just how much is necessary and how does it all fit together? From the VCFG and VSCL registers, you have defined a single scan line as having 256 pixels. (Actually, 256 colored pixels is beyond the bandwidth of NTSC. But let's not worry about that right now.) Each pixel will require 2 bits of data to determine which color (or shade of gray) will be displayed at that pixel location. This will require 512 bits of data per line (i.e., 256 pixels/line × 2 bits/pixel). A field has 262.5 scans lines that make up one screen scan. The first and last few are usually out of the field

Bits	Video scale register (VSCL)
31:20	N/A
19:12	PixelClocks (Number of CTRA PLL clocks/pixel) \$00-FF 8-bit value
11:0	FrameClocks (16 or 32 × PixelClocks) \$000-FFF 12-bit value

Table 2—This 32-bit register contains an 8-bit count of clocks per pixel and a 12-bit count of clocks per frame (1- or 2-bit resolution-dependent).

of view, leaving about 244 maximum viewable lines. Gamers will limit themselves to approximately 200 lines to be sure their environment is not chopped off at the top or bottom. So, at 200 scan lines, you need 12.8 KB of data space (512 bits/scan line \times 200 lines/8 bits). In many circumstances, you would like to use double buffering, which means two equal size video buffers. That would mean 25.6 KB of RAM. That sure doesn't leave much room for the application.

TILING

Earlier I mentioned that there are 256 character graphics stored in ROM. Each character is made up of 16 bits (horizontally) \times 32 bits (vertically). Two characters are combined to make use of the 32-bit double word data format. When the even or odd bytes of horizontal data are displayed on 32 separate scan lines, a picture of that character appears on the screen. If you want to print a character to the screen, you need a single byte to define the chosen character. The application doesn't need to figure out what data is required to form a character on the screen. All that is waiting for you to access it via a ROM address. This might require 1 byte pointer instead of 64 bytes of RAM (i.e., $16 \times 32 = 512$ bits).

Similarly, you can create special characters called tiles. A tile is used to create a background. You can think of a tile as a piece of a puzzle. The puzzle (picture) is a grid of horizontal and vertical positions on the screen where these pieces may be placed. The number of horizontal and vertical positions depends on the screen and tile resolutions. For instance, if each tile is 8 pixels \times 8 pixels and the screen is 256 pixels \times 200 pixels, then you can fit 32 tiles horizontally (256 pixels per scan line/eight pixels per tile) and 25 tiles vertically (200 lines/eight lines per tile). The tile bitmap of an 8 pixel \times 8 pixel tile would require 16 bits (eight pixels \times two color bits) per row times eight rows or 128 bits (four double words). For a gamer, the screen might be a bird's eye view of a maze. The entire picture could be drawn using only two tiles: a wall tile and a floor tile. Various mazes could be displayed by rearranging the two tiles in different patterns. Again, this reduces the amount of work associated with computing and storing a screen of information.

You can use tiles to dynamically change the way a screen is displayed; however, another object has been developed to operate in a more useful way. It is a sprite. While a tile and a sprite may have the same dimensions (and pattern), the latter has the ability to be placed anywhere on the screen and not just at the grid locations of tiles. One color of a sprite's pixel pattern is used as a transparent indicator that can let any tile color show through. The sprite can be magnified to become a multiple of its original size. And, most importantly, it has a depth associated with it that enables it to pass in front of or behind other sprites, creating a 3-D effect. While all of this is based on the tile/sprite generator written to run within a cog, this and many other Spin drivers written by the Parallax folks and other contributors like Andre LaMothe are available at www.parallax.com.

A BIT OF COLOR

If you have seen black and white TV, you may have been able to visualize color because your brain can take your



Photo 3—This photo of my VGA monitor's screen shows the simple display of three buttons with a horizontal gauge on a background of random characters and graphics. The red spot is the mouse cursor used to select a button. The buttons change a variable whose value determines the gauge's length. This data could come from an internal cog running a sampling application or from an external processor using the Propeller strictly as a display device.

real experiences and alter those sights broadcast in black and white. It was an extraordinary engineering feat to add color information to the standard black and white transmission signal without negatively affecting all the existing black and white receivers. A color sync signal hidden in the horizontal blanking portion of each scan line is disregarded by black and white sets, as is the modulated (and phase-shifted) color burst cycles during the active portion of each scan line. The average value left (of the color-modulated signal) remains as luminance levels for the black and white monitor producing levels of gray (between a black 0.25 V and white 1 V). A color monitor uses the phase difference between the color sync and the modulated color signal in the active portion of the scan line to determine color and the amplitude of the modulated signal to determine color saturation (or intensity). You saw earlier that the two video registers can configure the hardware to produce a composite video output signal containing all of the necessary syncs and modulations to do both black and white and color signal streams.

A cog driver can actually produce a signal with the full gamut of color. However, we've determined that it requires a lot of RAM to hold high-resolution color information. Because of the required RAM limits, RAM is conserved by limiting the resolution to 1 or 2 bits per pixel. One bit gives you black and white (or two colors). Two bits gives a few more colors to choose from, but how does this value relate to a specific color?

The grid that makes up the screen tile locations horizontal (rows) by vertical (columns) has a tile pointer map (TPM) associated with it. There is a 16-bit pointer for each tile location. Each pointer has double duty. The top 6 bits are an index into the tile color set table (TCT). The bottom 10 bits are an index into the tile bitmap memory (TBM). The tile color set table is a list of 64 4-byte entries. Each TCT entry holds the 8-bit color information for each of the four potential colors. Thus, this tile could choose to use any of the 64 color sets. The 10-bit

tile bitmap memory index is used as the upper 10 bits of a 16-bit address that holds all of the tile's bitmapped data.

GRAPHICS

While I consider characters, tiles, and sprites to be graphical in nature, the graphics engine driver uses point plotting to draw lines and polygons. You may be familiar with Cartesian coordinates, where x as a horizontal offset and y is a vertical offset from 0 in the center. Offsets increase when moving up and to the right, while they decrease when moving down and to the left of center. Any point P consists of an x and a y offset from 0. It is usually written as $P(x,y)$. Note that the video screen is usually mapped with an inverted y -axis so that $P(0,0)$ is the upper-left corner of the screen and $P(\text{screen_width}, \text{screen_height})$ is the lower-right corner of the screen. While the Cartesian coordinate system eases rendering, it is labor-intensive when dealing with rotations. Therefore, you may find the polar coordinate system more efficient when you must deal with rotational movements even though you will need to convert back and forth.

VGA

Up to this point, I've been discussing the power of the Propeller to work with NTSC video output. VGA video is actually less demanding than NTSC because the sync signal is digital in nature and separated from the color information. The color information is broken down into the three primary colors, and each has its own signal. The pixel clock is internally generated by the VGA monitor and will support 640×480 pixels. The VGA output consists of separated horizontal and vertical syncs plus separate R, G, and B analog outputs. Resister DACs can be used for each color similar to the DAC used for NTSC. Whereas the NTSC output requires 3 to 4 bits, the VGA output requires 8 bits. As for the VSU, it doesn't care which monitor is connected on the outside, as long as the timing configured into the video configuration registers is correct for the monitor type. **Photo 3** is an old VGA monitor (DB15 connection) I had connected to a Linux system here in the shop. It shows what can be done with just the Propeller demo board (or Hydra game console). If you do the math on 640×480 , you'll find that there isn't anywhere near enough RAM for this resolution, even at only 1 bit/pixel. However, by using 32×16 tiling, the RAM requirements are minimal.

I used the embedded character set to design a three-button screen with a linear bar graph. A mouse input enables button pushing, which in turn increases or decreases a variable that controls the length of the bar graph. The center button selects alternate color sets for the bar inside the graphs frame. Think of the variable associated with the bar's length as data coming through the USB port or other alternative connections, such as SPI, I²C, TTL serial, or a parallel port controlled by a spare cog.


While most of the examples in LaMothe's book use a composite NTSC output, you can find enough stuff inside about VGA to get started experimenting with some useful outputs. You can certainly start writing your own drivers for another microcontroller, but Propeller has some good things

going for it. With the internal PLL and an external 5-MHz crystal, the Propeller can clock at up to 80 MHz. With eight cogs, it's like having eight programs executing in parallel. Each cog has its own cog and also has its own VSU that makes outputting streaming video or audio a snap. While the resolution might be limited by the RAM available, the Propeller makes transitioning from NTSC to VGA a simple matter of software.

FARE THEE WELL

Our broadcasting buddy NTSC brought us closer to our world. We've seen world disasters, war, and poverty, as well as disaster relief, the Olympic games, and men landing on the moon. Thanks to NTSC, we've experienced positive and negative events together as one world. Digital broadcasting won't improve the standard of living for those in need, but it can carry on the tradition of NTSC by helping us understand more clearly (in HD) that we are no better than the least of our brothers.

And so we say goodbye to NTSC. It's been good knowing ya!

If you check your endangered species list, you might find that the CRT is hovering around the top. Not many bulky lead-shielded glass tube TVs (or computer monitors) are being manufactured. This is a case of less is more. LCDs have less weight and require fewer watts. Although we might see a continuing variety of interfacing connectors, for now all conform (for the most part) to the all-encompassing VGA (UXGA covers 1080p) standard. Who would've thought we'd have access to streaming TV programming via a cell phone? Cartoonist Chester Gould gave Dick Tracy the first two-way wrist radio in 1946, thanks to Al Gross's work on the walkie-talkie.^[2] 

Jeff Bachiochi (pronounced BAH-key-AH-key) has been writing for Circuit Cellar since 1988. His background includes product design and manufacturing. You can reach him at jeff.bachiochi@imaginethatnow.com or at www.imaginethatnow.com.

REFERENCES

- [1] D. Lancaster, "TV Typewriter," *Radio Electronics*, Gernsback Publications, New York, NY, 1973.
- [2] Winnipeg Free Press, "Born Too Soon," 2001, www.comsoc.org/socstr/org/operation/awards/assocpress.html.

RESOURCE

A. LaMothe, *Game Programming for the Propeller-Powered HYDRA*, www.parallax.com, www.xgamestation.com.

SOURCES

HYDRA Development kit
Nurve Networks | www.xgamestation.com

Propeller
Parallax | www.parallax.com