XAPP248 (v1.0) January 7, 2002

Digital Video Test Pattern Generators

Author: John F. Snow

Summary

This application note describes methods of efficiently generating standard video test patterns in Xilinx FPGAs. Video test patterns are used to verify the proper operation of video equipment. Most video equipment capable of generating a video signal can produce one or more video test patterns to verify proper operation of the video generator and attached video equipment. Thus, there is often a need to have a video test pattern generator embedded in the video equipment.

Two basic video pattern generator designs have been described in this application note. The first is based on distributed SelectRAM™ memory and is applicable to any current generation Xilinx FPGA family. The second design is based on the block SelectRAM memory in the Virtex™-II series. The design can implement sophisticated and flexible video pattern generation using very few Virtex-II device resources.

A Brief Component Digital Video Primer

Component Digital Video Standards

There are many different video standards, both analog and digital. Today, most broadcast studios and video production centers use component digital video when creating, storing, and transporting video. Component digital video can be readily compressed using digital video compression standards. It can also be encoded into analog composite video for broadcast.

Probably, the most common component digital video standards in use today are based on the 4:2:2 sampling scheme. The 4:2:2 component digital video format is used in various standards for 525-line (NTSC), 625-line (PAL), wide-screen NTSC and PAL, and HDTV. Table 1 lists some of the 4:2:2 component digital video standards.

Table 1: **Common 4:2:2 Component Digital Video Standards**

Standard	Description
SMPTE 125M(1) and ITU-R BT.601-5(2)	NTSC & PAL 4x3 aspect ratio 4:2:2 component digital video
SMPTE 267M	NTSC 16x9 aspect ratio 4:2:2 component digital video
SMPTE 260M	1125 Line 60-Hz HDTV
SMPTE 274M	1920 x 1080 Scanning – Progressive and interlaced HDTV
SMPTE 293M	720 x 483 Active – Progressive Scan HDTV
SMPTE 296M	1280 x 720 Active – Progressive Scan HDTV

Notes:
1. SMPTE - Society of Motion Picture and Television Engineers
2. ITU - International Telecommunication Union

The digital test pattern generators described in this application note are all designed to generate 4:2:2 component digital video. These designs are focused on standard definition video standards but are flexible enough in design to allow them to be modified to support HDTV standards.

Color Space

Black-and-white TV uses only intensity information, called luminance or luma designated with the letter Y. When color information was added, the luma signal was left intact for compatibility with existing equipment, and two components of color information, called U and V, were added. The two color components are often called color difference signals because they are derived by taking the difference between a color's intensity and the overall luminance of the sample. The U component is the difference between blue and Y. The V component is the difference between red and Y.

The PAL and NTSC TV broadcast systems are both based on the YUV color space. NTSC can also optionally use a derivative of YUV, called YIQ. "I" stands for in-phase and "Q" for quadrature, reflecting the modulation method used to transmit the color information.

The YCbCr color space is commonly used in component digital video. YCbCr is a scaled and offset version of the YUV color space with a luma component (Y) and two chroma (color difference) components (Cb and Cr). The Y component has a nominal 8-bit range of 16 through 235. The two chroma components have nominal 8-bit ranges of 16 to 240. Some values above and below the nominal ranges are used to encode special signals.

Sampling Schemes

One of the key characteristics of digital component video formats is the sampling scheme. Component video sampling schemes are denoted with a sequence of numbers separated by colons, such as 4:2:2 and 4:4:4.

A 4:2:2 sampling scheme indicates that for every four samples of luma (Y), there are two samples each of the two chroma signals (Cb and Cr). In standard definition video, the luma is sampled at a 13.5-MHz rate, while each chroma component is sampled at half that rate. This takes advantage of the fact that the human eye is less sensitive to color than to intensity to reduce the signal bandwidth by sampling color components at a lower frequency than the luma components.

Other common video sampling schemes are 4:4:4, where there are an equal number of Y, Cb, and Cr samples, and 4:1:1, where there is only one sample of each chroma signal for every four luma samples. A sampling scheme called 4:2:0 is often used in digital video compression standards and involves compression of the chroma components in both the horizontal and vertical direction rather than just the horizontal direction as is in 4:2:2. However, 4:2:2 is the most common sampling scheme in use today for component digital video in broadcast studios and video production centers.

For more detailed information on video sampling schemes, refer to **XAPP294: Digital Component Video Conversion 4:2:2 to 4:4:4.**

Video Format

For NTSC video, each video line contains 858 samples. As shown in Figure 1, a sample contains two words: a Y component word and a chroma component word, either Cb or Cr. Consecutive samples alternate between containing Cb or Cr components. The active video portion of the line consists of samples 0 through 719. The inactive portion or horizontal blanking interval of the line consists of samples 720 through 857.

For NTSC video, the four words of sample pairs 720/721 and 856/857 contain special codes called timing reference signals (TRS). The 720/721 pair contains the end of active video (EAV) TRS symbol, and the 856/857 pair contains the start of active video (SAV) TRS symbol. These TRS symbols are used to mark the transitions between the active and inactive portions of the line and also contain other timing information. When using 10-bit video words, the first three words of the TRS symbol are $3FF_{HEX}$, 000_{HEX}, and 000_{HEX}. The fourth word of the TRS symbol is called the XYZ word. Three bits of the XYZ word are used to indicate the status of the F, V, and H bits; four bits are used as error detection bits; and the remaining bits are fixed in value.

The F bit indicates whether field one (F = 0) or field two (F = 1) is active. The V bit is set to 1 in TRS symbols on lines that are part of the vertical blanking interval. On active video lines, the V bit is 0. The H bit distinguishes between EAV and SAV symbols. "H" is always a 1 in EAV symbols and always a 0 in SAV symbols.

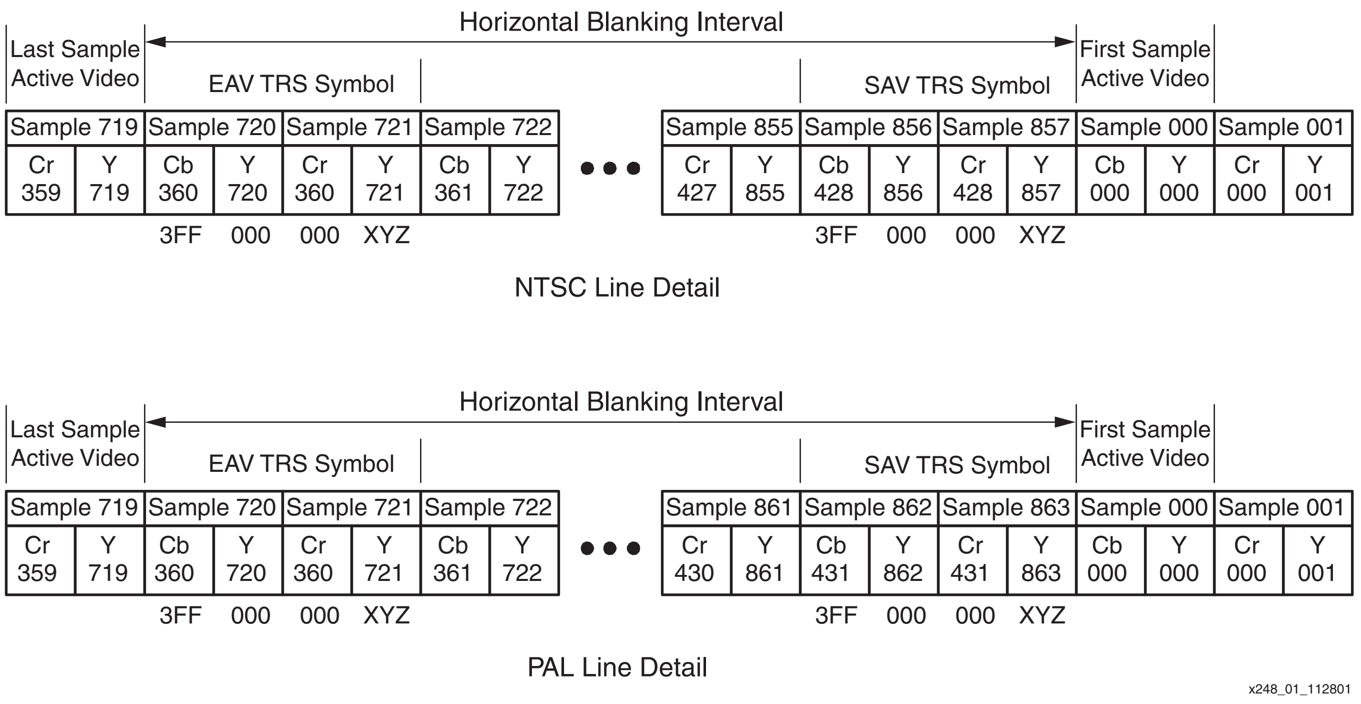
Horizontal Blanking Interval
Last Sample Active Video
EAV TRS Symbol
SAV TRS Symbol
First Sample Active Video
Sample 719 | Sample 720 | Sample 721 | Sample 722
Cr 359 | Y 719 | Cb 360 | Y 720 | Cr 360 | Y 721 | Cb 361 | Y 722
3FF 000 000 XYZ
Sample 855 | Sample 856 | Sample 857 | Sample 000 | Sample 001
Cr 427 | Y 855 | Cb 428 | Y 856 | Cr 428 | Y 857 | Cb 000 | Y 000 | Cr 000 | Y 001
3FF 000 000 XYZ
NTSC Line Detail
Horizontal Blanking Interval
Last Sample Active Video
EAV TRS Symbol
SAV TRS Symbol
First Sample Active Video
Sample 719 | Sample 720 | Sample 721 | Sample 722
Cr 359 | Y 719 | Cb 360 | Y 720 | Cr 360 | Y 721 | Cb 361 | Y 722
3FF 000 000 XYZ
Sample 861 | Sample 862 | Sample 863 | Sample 000 | Sample 001
Cr 430 | Y 861 | Cb 431 | Y 862 | Cr 431 | Y 863 | Cb 000 | Y 000 | Cr 000 | Y 001
3FF 000 000 XYZ
PAL Line Detail
x248_01_112801

Figure 1: **NTSC and PAL Video Line Detail**

The encoding of the TRS symbol's XYZ word is shown below:

Bit	9	8	7	6	5	4	3	2	1	0
	1	F	V	H	P3	P2	P1	P0	0	0

The bits labeled P3 through P0 are protection bits and are calculated in the following manner:

P3 = V XOR H

P2 = F XOR H

P1 = F XOR V

P0 = F XOR V XOR H

Figure 2 and Figure 3 show the arrangement of the vertical regions for both NTSC and PAL component digital video. The diagram shows the line numbers on which the F and V bits change values. For example, in NTSC video, lines 1 through 3 have both the F and V bits set to "1". On lines 4 through 19, the V bit is still a "1", but the F bit is a "0".

An NTSC video frame consists of 525 lines and is divided into two interlaced fields. Frames are drawn at a rate of 30 Hz. However, because new fields are drawn at a rate of 60 Hz, the flicker that the eye would perceive in a 30-Hz image is significantly reduced.

PAL video lines have the same number of active samples (720) as NTSC video. However, PAL has a few more inactive samples per line. PAL frames consist of 625 lines divided into two interleaved fields. The refresh rate of PAL is lower than NTSC with frames drawn at a 25-Hz rate (50-Hz field rate).

More detailed information about NTSC and PAL digital component video formats can be found in **XAPP286: Line Field Decoder**.

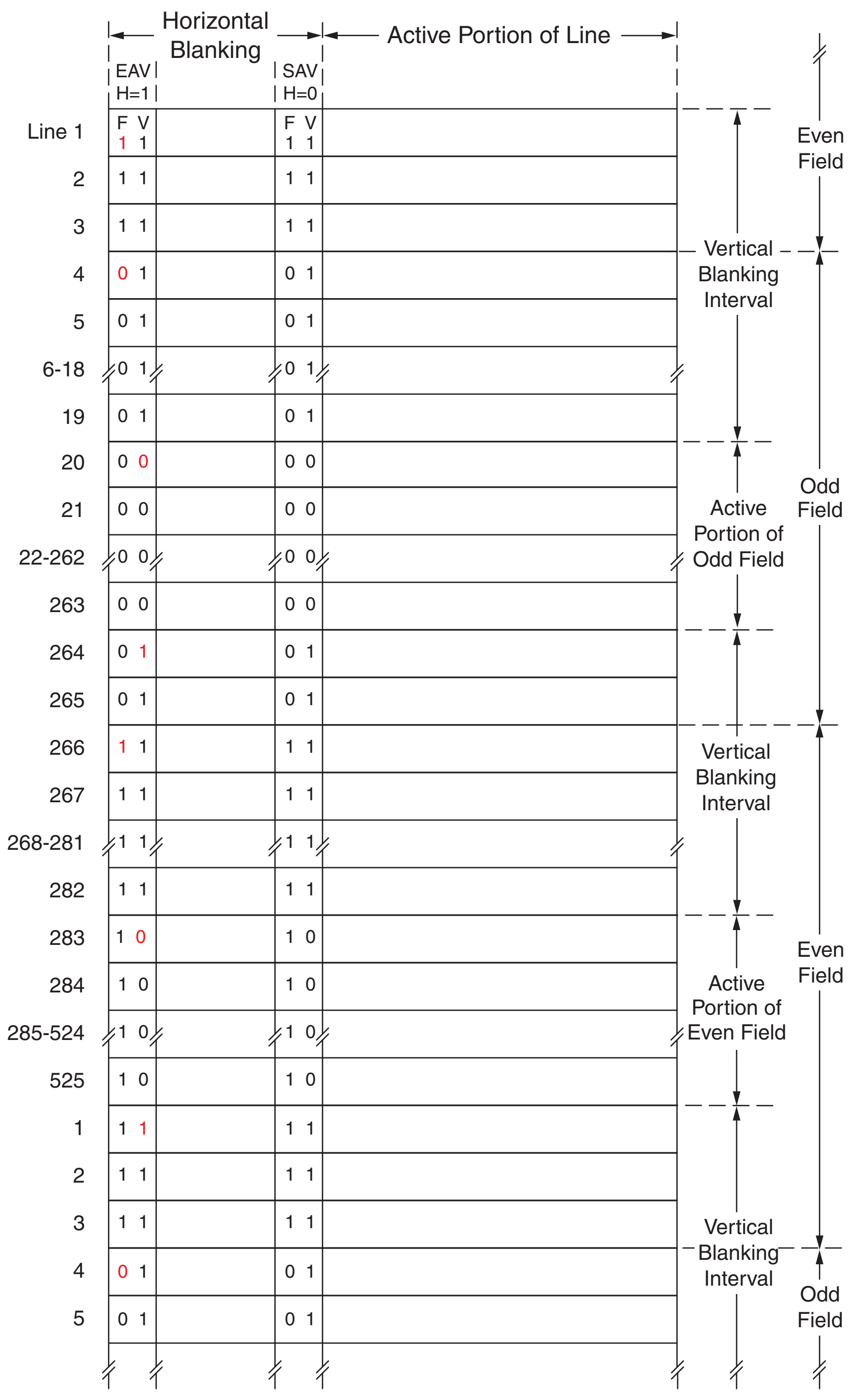

Figure 2: **NTSC Video Frame Details**

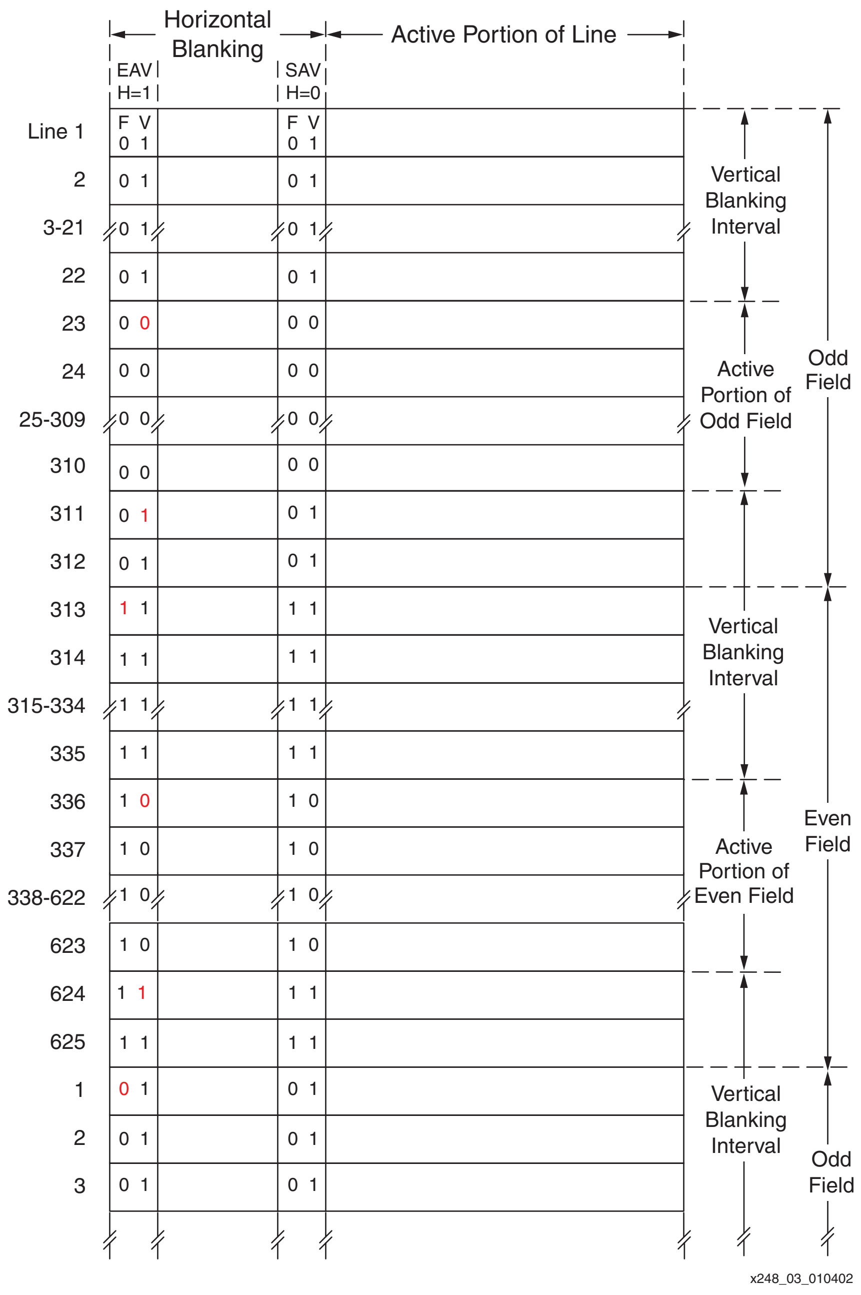

Figure 3: **PAL Video Frame Details**

Numbering Quirks

There are a few interesting quirks with the numbering of lines and samples in digital video. Lines are numbered beginning with one. In NTSC video, field 1 begins with line 4, not line 1. Field 1 includes lines 4 through 265 and field 2 includes lines 266 through 525 plus lines 1 through 3.

The samples along a horizontal line are numbered starting with zero. Sample 0 is the first sample of the active portion of the line. However, a new line does not actually begin at sample 0. Most digital video standards specify the beginning of the line as the first sample of the EAV symbol, sample 720. This is because the EAV symbol's V and F bits reflect the status of the line that follows, so it is convenient to think of the EAV symbol as being the beginning of the line.

In NTSC video, the definition of line 20 is somewhat unclear. Some documents define line 20 as the first active line of the odd field. Others define line 20 as the last line in the vertical blanking interval and line 21 as the first active line. Some of this confusion arises from the fact that the number of lines in the vertical blanking interval is given as a minimum of 19 lines in many NTSC standards, implying that it can be longer than 19 lines. If lines 1 through 19 are used as the vertical blanking interval, then line 20 is the first active line. Some prefer to use line 21 as the first active line because this gives an equal number of active lines (243) in each field. Using line 20 as the first active line gives 244 lines in the odd field and 243 in the even field.

Another element in this confusion is that some earlier versions of the NTSC digital component video standards ANSI/SMPTE 125M and ITU-R BT.656 allowed the V bit in the TRS XYZ word to transition from 1 to 0, indicating the end of the vertical blanking interval, on any line from 10 to 20 for the odd field and 273 to 283 for the even field. Current versions of these documents are now very precise in specifying that the V bit should be 1 on line 19 and 0 on line 20, making line 20 the first active line of the odd field.

Because of the ambiguity surrounding line 20, some video equipment manufacturers building NTSC digital video equipment treat line 20 as an active line, but avoid putting critical video information in the active video portion of line 20. The video test pattern generators in this application note all treat NTSC line 20 as a valid active line.

Video Test Pattern Standards

Standards for Color Bar Test Patterns

Many of the most commonly used video test patterns fall into the class called "color bars." Color bar test patterns consist of several vertical bars filled with primary and complementary colors. Color bar test patterns are particularly useful for verifying proper operation of video encoders and decoders and for adjusting video monitors.

One of the early color bar standards, traditionally called RS-189-A, is now officially called EIA-189-A.[Ref.3] Refer to Figure 4 for a diagram of the EIA-189-A color bar standard.

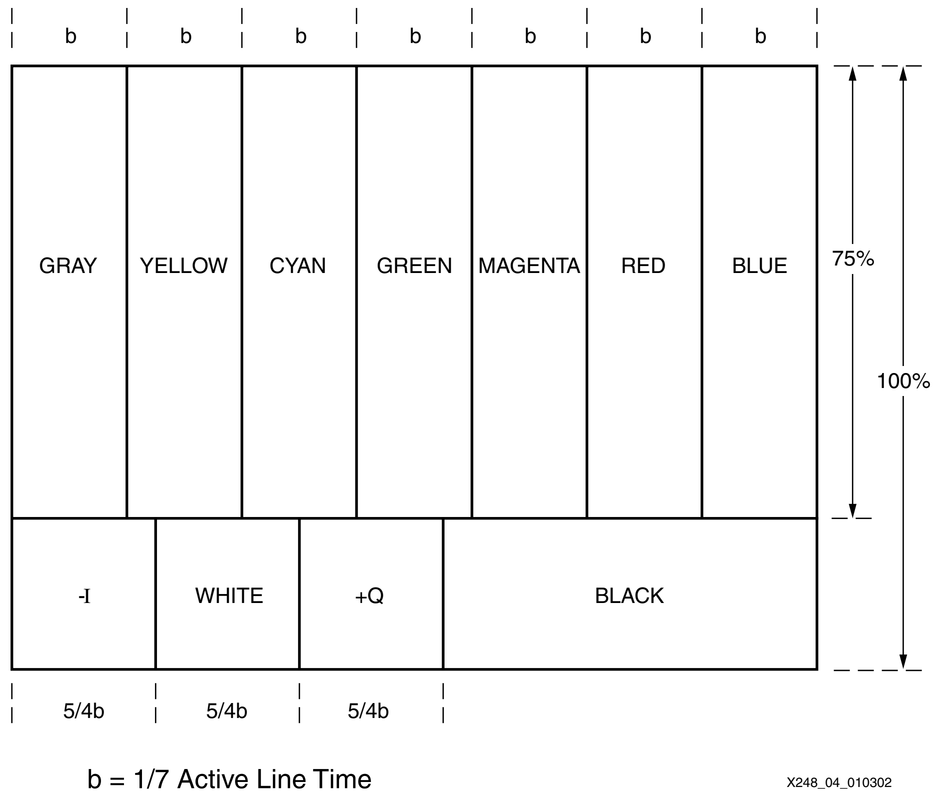

Figure 4: **EIA-189-A Color Bar Pattern**

The EIA-189-A test pattern consists of seven vertical color bars that occupy the top 75% of the test pattern. These color bars are called "75% bars," not because they occupy 75% of the picture, but because the luma value of these bars is set to 75% of the maximum luma value.

The bottom 25% of the test pattern consists of four bars of the colors –I, white, +Q, and black. The white bar is called 100% white because the luma component it set to 100%. The black bar is often called 0% black because the luma component is set to the black level or 0% luma. The –I and +Q colors represent full scale I and Q values in the YIQ color space. The –I color represents a signal with the maximum negative I value and a Q value of zero. The +Q signal represents a signal with a maximum positive Q value and an I value of zero.

SMPTE improved the EIA-189-A color bar pattern in engineering guideline, EG 1-1990. The SMPTE EG 1 test pattern is now one of the most commonly used video test patterns.

As shown in Figure 5, the EG 1 color bar pattern added to the EIA-189-A pattern a narrow middle band of color bars called the "new chroma set" bars. The new chroma set bars are arranged so that when the red and blue guns of a video monitor are turned off and only the blue gun is active, the brightness of each bar in the new chroma set should match the brightness of the 75% bar located immediately above it.

EG 1 also adds several narrow "near black" bars useful for setting the black level of monitors. These bars are sometimes called the PLUGE signal (Picture Line Up Generating Equipment). To adjust the black level of the monitor the brightness control is adjusted so that the black+4% (or whiter-than-black) bar is just visible but the black–4% (or blacker-than-black) bar is not distinguishable from the surrounding 0% black bars.

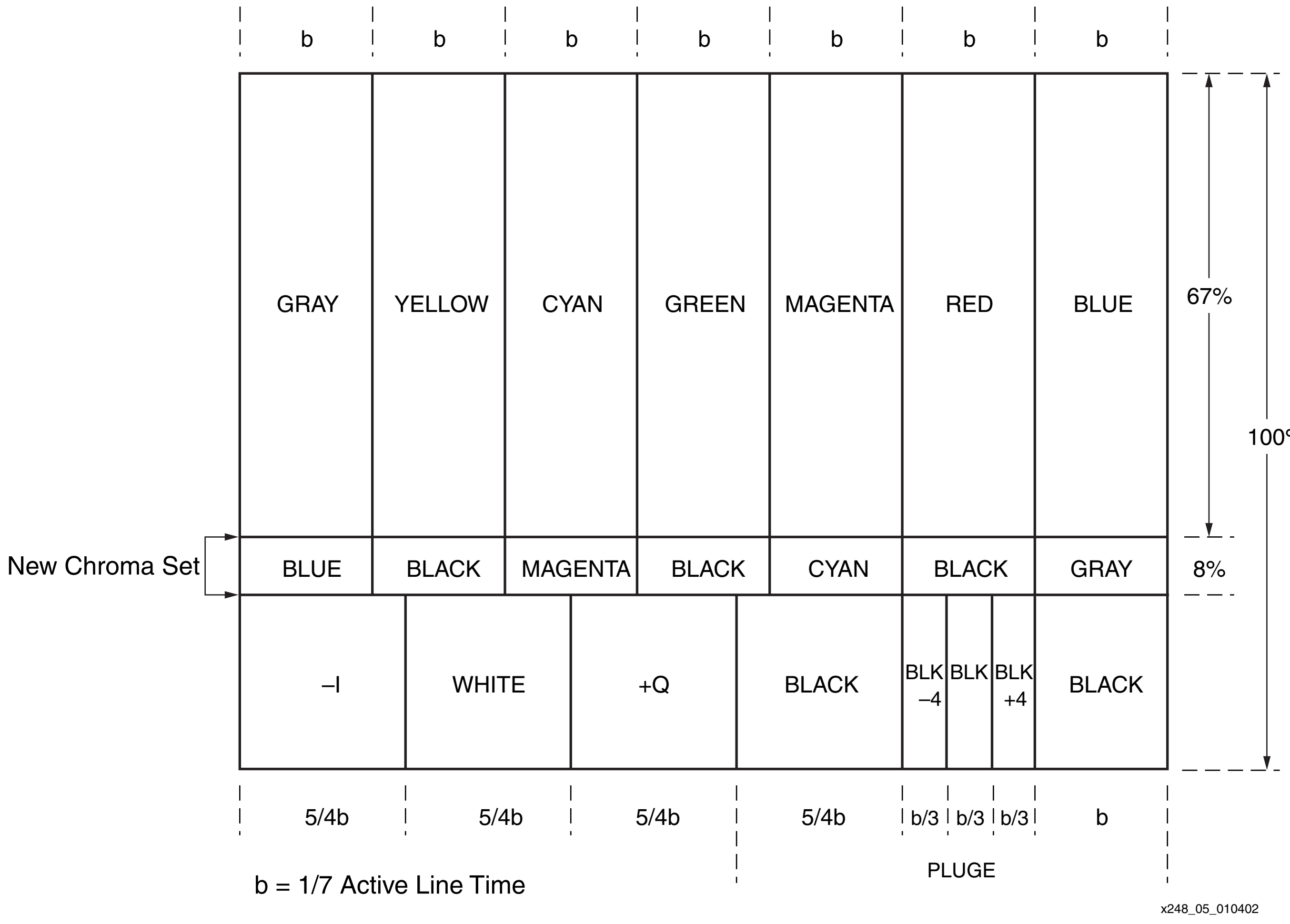

Figure 5: **SMPTE EG 1-1990 Color Bar Pattern**

SDI Pathological Test Patterns

Many video test patterns have been developed to aid in testing specific aspects of video equipment performance. An example of this is the SMPTE RP 178-1996 Serial Digital Interface (SDI) Checkfield.

Equipment complying with the SMPTE 259M SDI standard is widely used in broadcast studios and production centers to transport digital video over standard video coax cable. The SDI standard defines how to send digital video serially at bit rates ranging from 143 Mb/s to 360 Mb/s. To compensate for signal loss in the coax cable, the SDI standard requires adaptive cable length equalization at the receiver. This equalization circuit can be stressed by waveforms that have a high amount of DC content.

SDI receivers also require a clock and data recovery (CDR) circuit, usually based on a Phase Locked Loop (PLL), to recover the serial bitstream at the receiver. The CDR circuit requires bit transitions periodically to stay locked to the bit rate of the bitstream. Low frequency waveforms with long runs of 1s or 0s stress the CDR's ability to stay locked when few transitions are present in the bitstream.

The SMPTE recommended practice RP 178-1996 defines two test patterns, one to test the receiver equalization by producing a bitstream with a maximum amount of DC content and another to test the CDR circuit's low frequency response by producing a bitstream with long runs of 1s or 0s. The SDI "checkfield", as the RP 178 test pattern is called, has a cable equalizer test pattern in the first half (top) of each active video field and a CDR test pattern in the second half (bottom) of each active video field.

In the cable equalizer test pattern, all chroma (Cb and Cr) components have values of 300_{HEX} (all values are 10-bit values) and the luma (Y) components have values of 198_{HEX}. This pattern, when encoded by an SDI encoder, occasionally generates a repeating serial pattern that has 19 High bits followed by one Low bit or 19 Low bits followed by one High bit. This pattern produces a maximum amount of DC offset. To insure that both polarities of this pattern are generated, the entire video frame must have an odd number of 1 bits at the input to the SDI encoder. This is done by setting the Y component of the last sample on the first active line of the first field (line 20 for NTSC or line 23 for PAL) to a value of 080_{HEX} instead of 198_{HEX}.

In the CDR test pattern, all chroma components have a value of 200_{HEX} and all luma components have a value of 110_{HEX}. Feeding this pattern into an SDI encoder for one-half of a field produces several lines of a repeating waveform that has 20 consecutive bits of one polarity followed immediately by 20 consecutive bits of the opposite polarity, producing a minimum number of transitions to the CDR circuit.

The SDI encoding scheme uses a linear feedback shift register (LFSR) to scramble the video data. The starting state of the LFSR affects how any 10-bit video word is encoded by the SDI encoder. An SDI encoder has 511 different possible starting states. When the RP 178 patterns are encoded by an SDI encoder, they do not immediately nor consistently generate the pathologic waveforms. They only generate the pathological waveforms once the encoder has reached a certain starting state. In the half field where each of the two test patterns is applied to the SDI encoder, the pathological waveforms are only generated by the encoder during a few of the active video lines.

Reference Designs

Two basic video pattern generator designs are presented here, with a few minor variations of each design type also provided. The first type of pattern generator is based on distributed RAM found in most Xilinx FPGA families. The second basic type is based on the Virtex-II synchronous 18K-bit block RAM.

Limiting Signal Transition Rates

Many video standards require the video component values to have limited transition rates. This is because analog video devices have a limited amount of bandwidth and cannot handle video signals that transition quickly from one value to another. Color bar patterns have high transition rates at the borders between adjacent color bars.

A test pattern generator can limit the signal transition rates by ramping the value of the digital components at the color bar transitions. However, since video encoders often limit signal transition rates, it is often easier to let the video encoder perform this function.

The test pattern generators described in this application note do not limit the transition rates of the signals. If transition rate limiting needs to be implemented in the FPGA, this can be done with a video FIR filter connected to the output of the test pattern generator. The FIR filter function must only be applied to the active video data and not to the timing reference signals or any non-video digital data that may be included in the blanking intervals. The design of a video FIR filter is outlined in other Xilinx applications notes.

Some test patterns, such as the RP 178 SDI test patterns, must not be filtered. Filtering the RP 178 patterns prior to SDI encoding does not achieve the correct test effect.

Distributed RAM Video Pattern Generators

Figure 6 provides a block diagram of a color bar pattern generator based on ROMs implemented in distributed RAM. This pattern generator produces the SMPTE EG 1 color bar pattern. The pattern generator can be broken down into three main sections: the horizontal section, the vertical section, and the component video generator section.

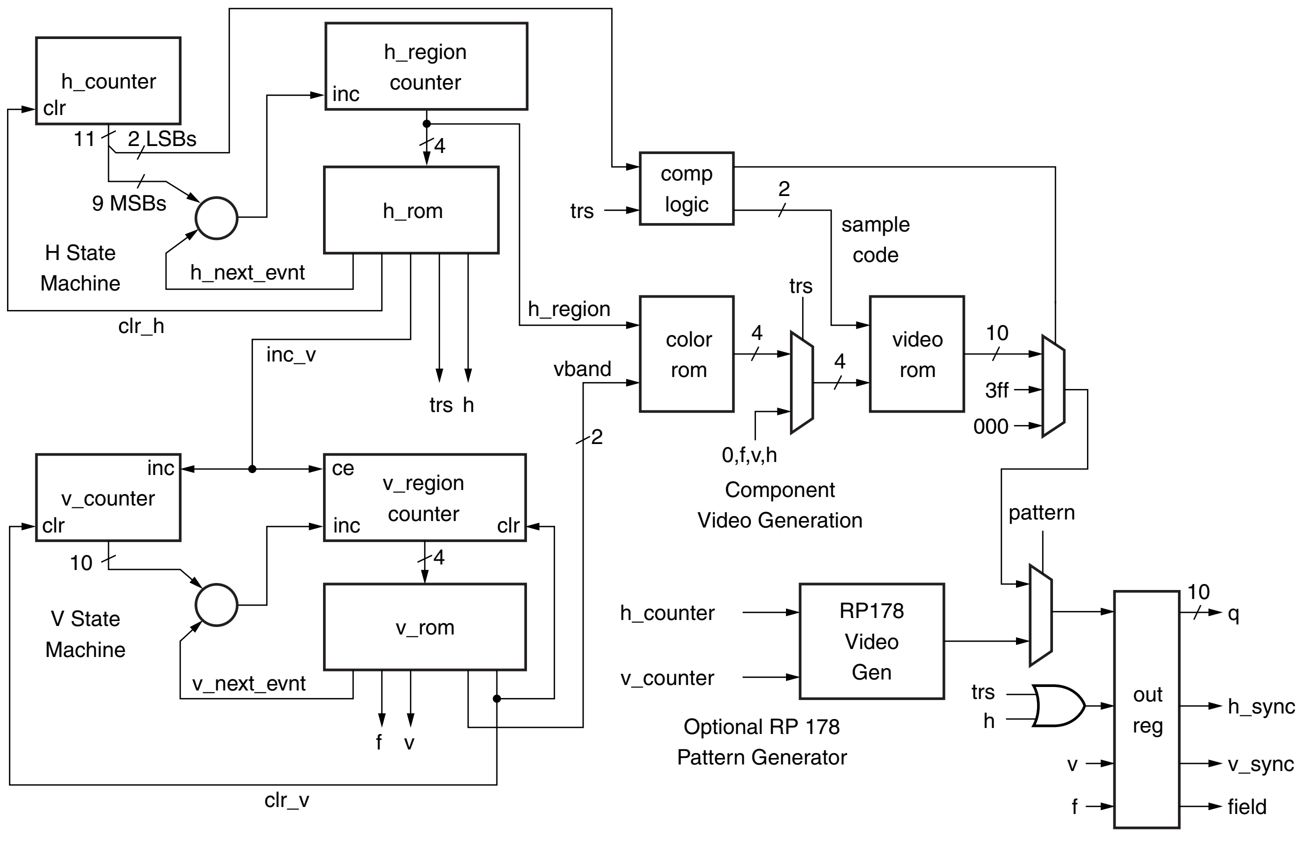

Figure 6: **Distributed RAM EG 1 Pattern Generator**

Horizontal Section

The horizontal section contains a horizontal counter and a horizontal state machine. The horizontal counter increments every clock cycle, counting the number of words (two words per video sample) on a horizontal video line. The two least significant bits (LSBs) of the horizontal counter are used to determine which component to output: Y (01 or 11), Cb (00), or Cr(10). The horizontal counter is reset to zero by the horizontal state machine when the end of the video line is reached.

The horizontal state machine sequences through a series of horizontal regions on each video line. The transition from one horizontal region to another is called a horizontal "event." A horizontal event must be defined at each possible point on the line where the outputs of the horizontal state machine must change. These events occur where a new color bar could begin or a TRS symbol must be generated. In this design, horizontal events can only occur at four clock boundaries. That is, horizontal regions can only begin where the least significant two bits of the horizontal counter are both zero.

Figure 7 shows the EG 1 color bar pattern. Along the bottom of the drawing, the horizontal events and regions are defined. Each horizontal event is marked by a dotted line. The horizontal counter value for the beginning of each horizontal region is also shown. These counts are valid for NTSC video. Note how some color bars span multiple horizontal regions. For example, the top red color bar spans three regions because of the three small PLUGE bars located below it.

The last horizontal region on the right of the pattern is defined as just the last two samples of the line and is the horizontal region where the horizontal state machine asserts a signal to cause the vertical state machine to increment the line counter.

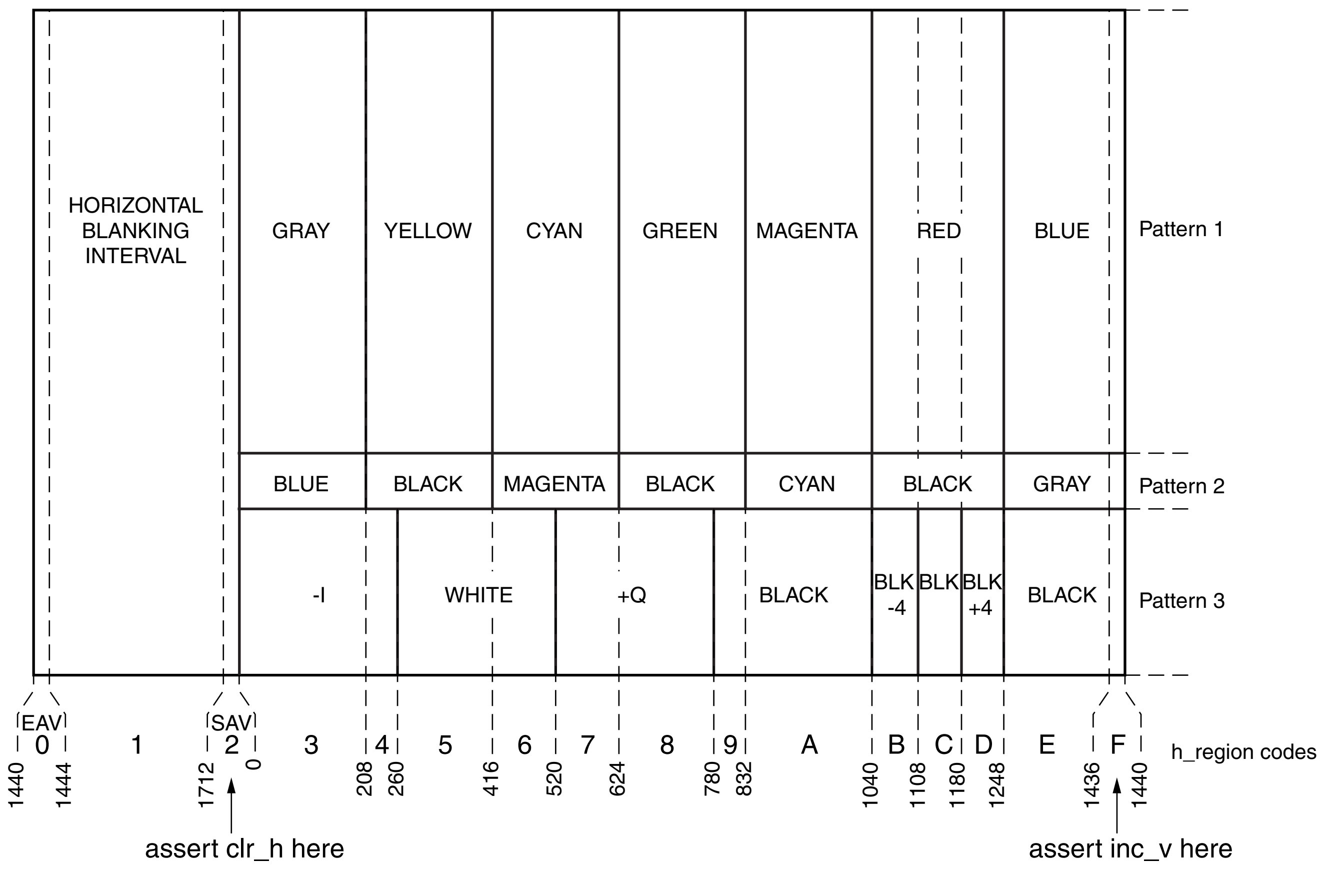

Figure 7: **Horizontal Regions of the EG 1 Test Pattern Generator**

The horizontal state machine consists of a horizontal region counter containing the current state (horizontal region) of the state machine and a ROM to decode the current state into the control outputs. One of the outputs of the ROM is a 9-bit "next-event" value. A comparator constantly compares the most significant nine bits from the horizontal counter to the next-event value from the ROM. When they match, an event has been reached and the horizontal region counter is incremented, moving the horizontal state machine to the next horizontal region. The horizontal region counter will only increment when the least significant two bits of the horizontal counter are both 1s.

The horizontal ROM also generates several other outputs:

clr_h clears the horizontal counter when the end of the line is reached

inc_v causes the vertical counter to increment to the next video line

trs indicates that a TRS symbol will be generated in the current horizontal region

h H bit (horizontal blanking indicator) for the TRS XYZ word

Vertical Section

The vertical section contains a vertical counter that keeps track of the current line number. It increments from one to the maximum number of lines in the frame. The horizontal section controls when the vertical counter increments by asserting the *inc_v* signal at the end of each horizontal line. The vertical counter is cleared to a value of 1(remember that the first video line is 1) when the vertical state machine asserts the *clr_v* signal indicating the end of the frame.

The vertical section also contains a vertical state machine that is almost identical to the horizontal state machine. A vertical region counter contains the current vertical region value.

A ROM decodes the vertical region value into a number of control bits, including a 10-bit next-event value that is constantly compared to the current value of the vertical counter. When the current line number matches the next-event value from the ROM, the vertical region counter is incremented. The vertical region counter only increments at the beginning of the video line as indicated by the horizontal state machine asserting the *inc_v* signal.

Different vertical regions are required to keep track of the changes in the V and F bits and for the different vertical patterns in the EG 1 pattern. Figure 8 shows an NTSC video frame with the 11 different vertical regions that the vertical state machine cycles through to process one frame of video. Note that the last region, region 10, is only active for the last video line. During this region, the *clr_v* signal is asserted to cause the vertical counter and the vertical state machine to reset to the beginning of the frame when the end of the line is reached.

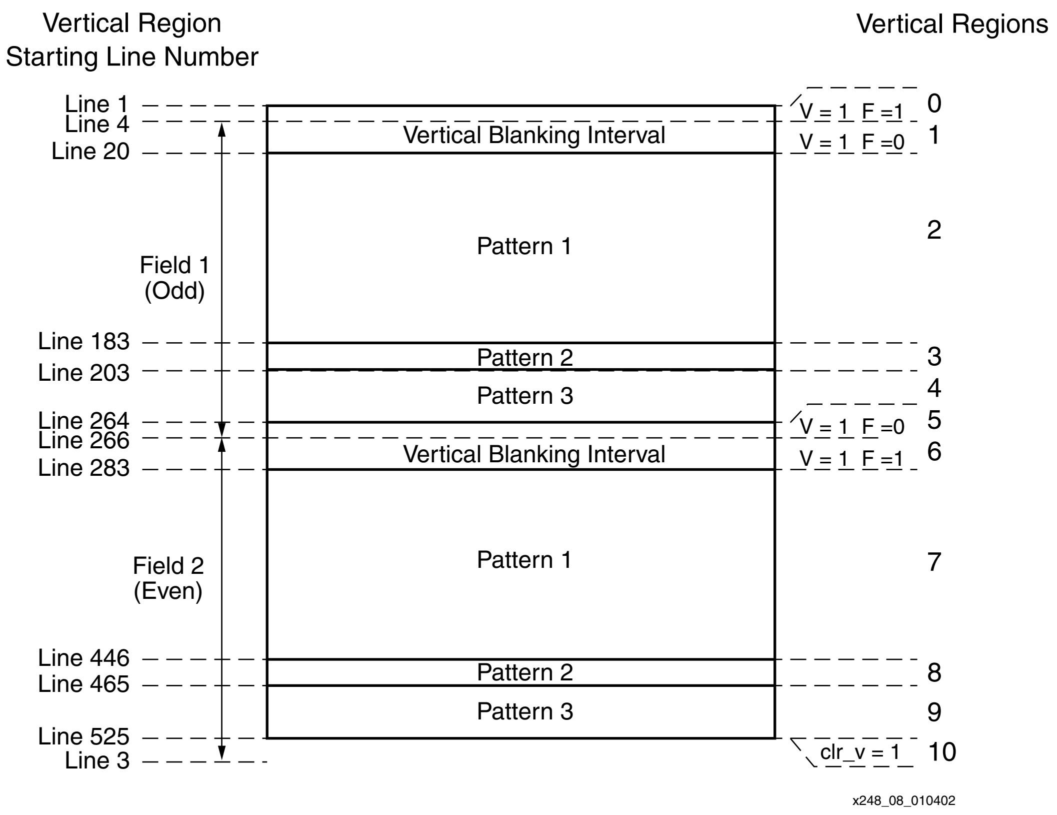

Figure 8: **Vertical Regions of the EG 1 Test Pattern Generator**

The outputs of the vertical state machine are:

f is the F bit (field indicator) for the TRS XYZ word

v is the V bit (vertical blanking indicator) for the TRS XYZ word

clr_v clears the vertical counter and the vertical region counter at the end of the video frame

vband indicates which vertical region is currently active

The two-bit *vband* signal indicates which of the three patterns (color bar sets) should be generated, based on the current vertical position. The EG 1 test pattern has three color bar sets located in three different rows on the screen The fourth value that *vband* can assume indicates that the current vertical region is a vertical blanking interval.

Component Video Generator Section

The component video generator section converts the 2-bit sample code from the horizontal counter, the 4-bit horizontal region code, and the 2-bit *vband* code into actual video component values. Two ROMs are used in the component video generator section.

The color ROM converts the *vband* and horizontal region codes into a 4-bit color code. The EG 1 test pattern uses 13 different colors, leaving three unused color codes.

When generating colors, not TRS symbols, the 4-bit code from the color ROM and a 2-bit sample code derived from the two LSBs of the horizontal counter form the address into the video ROM. The video ROM generates the 10-bit value for each component of the color.

The sample code tells the video ROM whether to output the Y, Cb, or Cr components of the color. This uses three of the four possible values of the sample code. The fourth value of the sample code indicates that a TRS symbol should be generated. When the horizontal state machine asserts the TRS signal, a MUX located between the color ROM and the video ROM replaces the color code from the color ROM with the F, V, and H bits. The video ROM encodes the F, V, and H bits into a 10-bit XYZ word for the TRS symbol.

The video ROM only generates the XYZ word of the TRS symbol. It does not generate the first three words of the TRS symbol. The values of these first three words are $3FF_{HEX}$, 000_{HEX}, and 000_{HEX}. These values are generated by a MUX on the output of the video ROM. When a TRS symbol is being generated, the MUX supplies 3FF for the first word, 000 for the second and third words, and the video ROM supplies the XYZ word for the fourth word. The use of a MUX to generate the trivial 3FF and 000 values reduces the amount of space needed in the video ROM.

The EG 1 color bar generator using distributed RAM is implemented in the **cb_eg1.v** and **cb_eg1.vhd** files. When generating NTSC or PAL video using this design, a 27-MHz clock should be used.

Generating the RP 178 SDI Checkfield

The RP 178 SDI checkfield pattern is relatively simple. It consists of one pattern during the first half of the active field and another pattern during the second half. However, there is one exception where the last Y component on the first active line of the first field has a different value than the other Y components in the cable equalization pattern.

In the **cb_eg1_rp178**.* files, an RP 178 pattern generator has been grafted onto the EG 1 color bar generator described above. An input signal to this module indicates whether the EG 1 or RP 178 pattern should be generated. The RP 178 generator simply looks at the horizontal and vertical counters to determine which video component values to output. This RP 178 generator only generates values during the active portion of the video. The regular color bar pattern generator takes over and generates the TRS symbols and blanking interval values.

A reference design that generates only the RP 178 SDI Checkfield test pattern is provided in the **rp178.v** and **rp178.vhd** files. This design is based on the distributed RAM test pattern generator design, but only generates the RP 178 test pattern, making it smaller than the combined EG 1 and RP 178 test pattern generator.

Simple Color Bars

The HDL files **colorbars.*** contain a simplified version of the EG 1 color bar generator. This version simplifies the bottom pattern so that a gray bar occupies the left half of the pattern and black bar occupies the right half. This eliminates the need to generate the colors –I, +Q, white, and the two near-black signals. This reduces the number of colors needed from 13 to 8, allowing the color code generated by the color ROM to be reduced from four to three. These changes reduce the size of the video ROM to half and eliminate one bit from the color ROM, resulting in a smaller implementation.

This simplified version can be used when space is at a premium in the FPGA and strict adherence to the EG 1 standard is not required.

Block RAM Video Pattern Generators

The dual-port Virtex-II block RAMs allow two independent test pattern generators to be implemented, using the same amount of hardware as required to implement one pattern generator. The second generator is essentially free. The two pattern generators must share the same ROM data, meaning they generate the same patterns but do not have to be synchronized in any way. An EG 1 color bar generator can be made using three Virtex-II block RAMs and very few other FPGA resources.

Figure 9 is a block diagram of a block RAM-based video pattern generator. The block diagram shows only one pattern generator, but this design implements two independent pattern generators in three block RAMs and four Virtex-II slices. If there are three unused block RAMs in a design, a video pattern generator can be added for almost no additional cost.

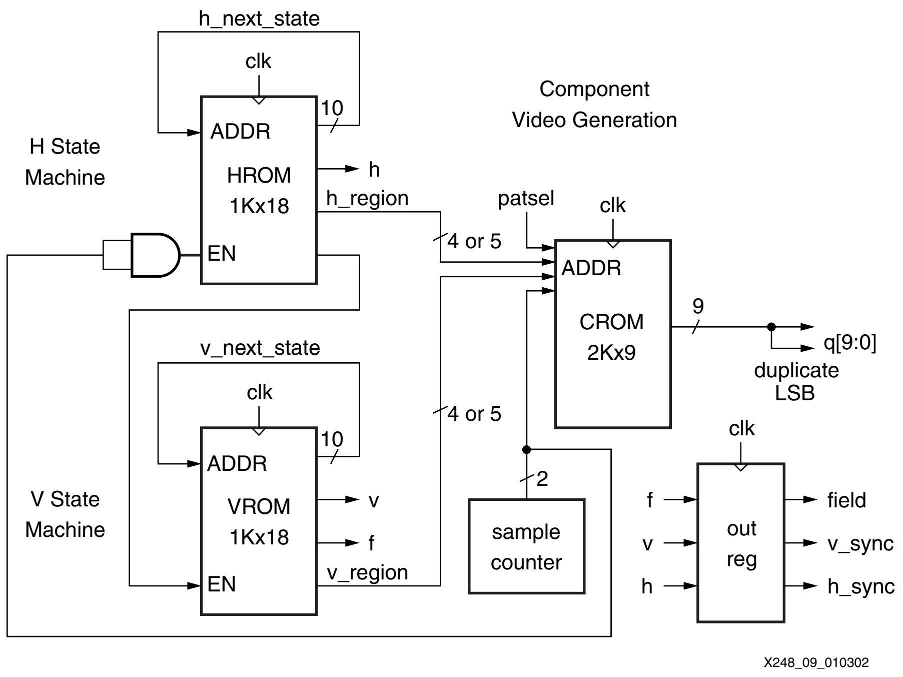

Figure 9: **Video Pattern Generator Using Block RAMs**

HROM

The HROM is a block RAM configured as a 1Kx18 device. It implements the horizontal state machine with the internal register of the block RAM serving as the current state register. Ten bits out of the HROM form the "next-state" value and wrap back around to the address input of the HROM. The HROM state machine advances one state every four clock cycles. A 2-bit sample counter is decoded to provide the clock enable signal for the HROM.

The HROM can implement up to 1024 states. Because each state lasts for two video samples, the HROM can accommodate test patterns that are up to 2048 samples wide, sufficient to cover most standard definition video formats. Some wide-screen standard definition video formats and some high-definition video formats have more than 2048 samples per line. There are two ways to adapt the design for these HDTV video formats. First, the number of samples per state could be increased to four, providing for up to 4096 samples. Doing so would require some changes to the design to correctly generate the TRS symbol during half of the state. Second, an additional HROM could be added to expand the HROM to 2Kx18, providing twice as many states.

The HROM generates an *h_region* code to indicate which horizontal region is currently active. The *h_region* code can be either four or five bits wide, depending on the requirements of the test pattern.

The HROM asserts a signal called *h* during the horizontal blanking interval. It also generates an enable signal to the vertical state machine. This enable signal indicates the end of the current line and causes the vertical state machine to increment to the next line when asserted.

VROM

The VROM is another 1Kx18 block RAM used to implement the vertical state machine. It is configured just like the HROM state machine with ten bits out of the VROM forming the next state value and wrapping back around to the VROM's address input.

The VROM can implement up to 1024 states. With each state corresponding to one video line, the VROM has enough states to cover most current video resolutions, but does not cover the 1080-line HDTV standards. To adapt this design to cover the higher resolution standards, the VROM can be implemented in two block RAMs each configured as 2Kx9, giving a total RAM space of 2Kx18.

The VROM generates a *v_region* code to indicate the current vertical region. The *v_region* code can be four or five bits wide. The VROM also generates a field indicator bit (*f*) and a vertical blanking indicator bit (*v*).

CROM

The CROM is a third block RAM in a 2Kx9 configuration and is used as the video component generator. The address inputs for CROM come from the 2-bit sample counter, the *h_region* code from the HROM, and the *v_region* code from the VROM. If both *h_region* and *v_region* are four bits wide, then there is an extra address input to the CROM available. This design example takes advantage of this extra address pin as a pattern select input, allowing either the EG 1 or RP 178 test pattern to be selected.

With the two independent test pattern generators available due to the dual-port nature of the block RAM, one generator can be generating the EG 1 pattern while the other is generating the RP 178 pattern. Or, they can both be generating the same pattern.

The CROM has a 9-bit wide output, so it can only produce 9-bit color components. While this is generally sufficient for most color bar applications, the RP 178 test patterns require all components to be generated at 10-bit resolution. TRS symbols should also be generated accurately to 10-bit resolution. There are several ways to solve this problem.

First, the CROM can be configured as a 1Kx18 part, allowing for more output resolution. This would limit the *v_region* and *h_region* codes to 4-bit values and would eliminate the ability to put two patterns in the CROM.

Second, an additional block RAM can be used to double the number of bits out of the CROM. Since many applications only require 10-bit video, it seems a waste to use an entire block RAM to generate one more bit.

Because generating video test patterns as efficiently as possible was a goal of this reference design, another technique was used. The LSB from the CROM is duplicated and used as both of the two LSBs of the component value. This can produce color component values that may be 1-bit different than recommended by some standards. However, it does accurately generate all TRS symbol words, and it also correctly generates all RP 178 test pattern component values.

A 3-bit output register is used to delay the *f*, *v*, and *h* bits from the VROM and HROM by one clock cycle to match the clock cycle of delay in the CROM.

When generating NTSC or PAL video using the **vidgen** design, a 27-MHz clock should be used.

Generating the ROM Contents

Using block RAMs as the basis for a video test pattern generator makes a very flexible design. The test patterns can be changed simply by changing the initialization values of the RAMs, or by reloading the RAMs on the fly. The difficult part is coding the contents of the these large RAMs by hand.

As part of the reference design, a utility called **cbgen** has been provided. This utility reads a text file that describes the test pattern and generates initialization files for the three RAMs. Two initialization files are generated for each RAM, one containing the initialization code for simulation and the other containing the synthesis initialization code. This utility generates the initialization files in either VHDL or Verilog and will produce correct synthesis code for XST, Leonardo, FPGA Express, or Synplify.

The utility can also generate files compatible with the Xilinx XDL tool, which allows the initialization values of the ROMs to be changed without resynthesizing or rerouting the FPGA design. Refer to the *cbgen User Guide* for a complete description on how to use the **cbgen** utility.

The **vidgen.v** and **vidgen.vhd** files contain the HDL descriptions of the block RAM-based video pattern generator. The **vidgen.v** files contain the ‘include directives that include the six RAM initialization files. Some Verilog synthesis tools do not implement the ‘include directive. In these cases, the initialization files should be inserted directly into the **vidgen.v** file where the ‘include directives currently exist.

VHDL lacks a file include directive, so the ***.vhd** initialization files should be inserted directly into the **vidgen.vhd** file at the places indicated by the comments.

The supplied RAM initialization files generate both the EG 1 and RP 178 test patterns for 4x3 aspect ratio, 525-line NTSC video. A pattern definition file that can be processed by **cbgen** is also provided for the EG 1 and RP 178 test patterns in 625-line PAL format.

The **cbgen** utility is provided pre-compiled and ready to run on a PC under Windows. The C source code for **cbgen** is also provided so that it can be compiled for use under other operating systems.

Reference Design Results

Table 2 shows the results after "place and route" of the various modules implemented in this application note. All results were obtained using the Verilog versions of the designs with Xilinx ISE version 4.1i using XST as the synthesis tool. Results using the VHDL files are not shown, but are essentially identical. Virtex-II device results are for a –5 speed grade device. Spartan-II device results are for a –6 speed grade device.

Table 2: **Reference Design Results**

Design Name	Optimized for Area			Optimized for Speed		
	Size LUTs/FFs	Speed Virtex-II Device	Speed Spartan-II Device	Size LUTs/FFs	Speed Virtex-II Device	Speed Spartan-II Device
colorbars.v	116/42	100 MHz	60 MHz	117/46	140 MHz	80 MHz
cb_eg1.v	132/42	100 MHz	60 MHz	137/60	140 MHz	80 MHz
cb_eg1_rp178.v	160/43	90 MHz	60 MHz	171/47	140 Mhz	80 MHz
rp178.v	82/41	140 MHz	90 MHz	86/41	165 MHz	100 MHz
vidgen.v	6/10	175 MHz	NA	6/10	200 MHz	NA

Reference Design Files

The reference design files can be downloaded from the Xilinx FTP site under xapp248.zip.

Conclusions

Video test pattern generators are often included in many types of video equipment, sometimes to provide a quick go/no-go test to determine if the equipment is functional, other times to provide sophisticated diagnostic capabilities.

Xilinx FPGAs are now commonly used in video equipment, so there is a need to efficiently implement video test pattern generators in Xilinx FPGAs. Two basic video pattern generator designs have been described in this application note.

References

The following references are recommended:

1. All of the SMPTE standards referenced in this application note are available from The Society of Motion Picture and Television Engineers and can be purchased at the SMPTE web site: http://www.smpte.org
2. The ITU-R BT.601-5 standard can be purchased from the International Telecommunication Union at: http://www.itu.int/itudoc/itu-r/rec/bt/
3. The EIA-189-A standard can be purchased from the Electronic Industries Alliance at: http://www.eia.org

Appendix A User Guide - cbgen

Introduction

The **cbgen** utility was developed to make it easier to generate the block RAM initialization files for the three ROMs in the **vidgen** video pattern generator reference design in Xilinx application note XAPP248. The utility reads a text file that describes the video test pattern and generates two initialization files for each of the three ROMs, one for simulation and one for synthesis.

The utility generates the initialization files in either VHDL or Verilog or in a format compatible with the Xilinx XDL tool. XDL allows the ROM initialization values to be modified after synthesis and place and route.

The utility is written in C and the source code is provided to allow the utility to be modified or to be compiled for different operating systems.

This version of **cbgen** is limited to generating initialization files for designs with only one block RAM per section. It cannot support video formats with more than 1023 lines or 2048 samples per line.

Input File Format

Basic Syntax

The **cbgen** input file is a text file that describes the video test pattern to be generated.

The file may contain comments that will be ignored by **cbgen**. The comment character is // and may occur anywhere on a line. Anything to the right of the comment character will be ignored. Blank lines are ignored.

Generally, **cbgen** expects a command to exist on a single line of the text file. However, there is a line continuation character: \. Anything on a line to the right of the line continuation character will be ignored. The following line is appended to any line with a line continuation character.

Some commands require an item to be named, such as the color definition lines in a PALETTE block. Names must only be a single word so they may not contain spaces.

The different elements on a command line must be separated by one or more space or tab characters.

File Sections

The input file is divided into different sections. The sections must appear in the proper order.

The first section contains a number of different commands that establish various parameters. These parameters include the number of words per video line, the number of lines in the video frame, and the names of the output files.

Next, the color palettes are defined using the PALETTE block. This block defines the colors that will be used in the video test pattern.

The HORIZONTAL_REGIONS block must come after the palette blocks. This block describes where each horizontal region in the test pattern begins and ends on a video line.

The LINE_FORMATS block must come after the horizontal regions block. This block defines the possible formats that a video line may have. It defines what colors should be generated in each horizontal region for different line types. A different type of line (or format) occurs in each vertical region. For example, in the EG 1 test pattern, the lines in the 75% color bars pattern at the top of the screen have a different format than either the lines in the middle "new chroma set" pattern or the lines in the bottom pattern.

The last section of the input file is the VERTICAL_REGIONS block. This block is where the extent of each vertical region is defined. It also associates the lines within each vertical region with a line format.

Parameter Commands

At the beginning of the file, there must be several single-line parameter commands. These commands may come in any order relative to one another. These commands are described below.

PALETTES num

This command specifies how many color palettes will be used in the test pattern. The numeric parameter may be either 1 or 2. The number of palettes used is usually one, unless two different test patterns are to be stored with a pattern select bit used to select between them. The pattern select bit is an extra address bit to the CROM and essentially selects between the two possible color palettes.

Only one color palette may be used if either the hregion or vregion codes are five bits wide.

If the PALETTES command does not appear in the pattern definition file, the number of palettes will default to 1.

HREGION_BITS num

This command specifies how many bits are used to encode the horizontal region. The numeric parameter may be either 4 or 5. Four bits allows 16 horizontal regions to be defined and five bits allows 32 horizontal regions to be defined. In determining how many horizontal regions to use, be sure to note that three horizontal regions are consumed by the EAV, horizontal blanking, and SAV regions. If five bits are used for the horizontal region code, only four bits may be used for the vertical region code and only one color palette may be used.

If the HREGION_BITS command does not appear in the pattern definition file, the number of horizontal region code bits defaults to 4.

VREGION_BITS num

This command specifies how many bits are used to encode the vertical region. The numeric parameter may be either 4 or 5. Four bits allow 16 vertical regions to be defined and five bits allow 32 vertical regions to be defined.

If the VREGION_BITS command does not appear in the pattern definition file, the number of vertical region code bits defaults to 4.

H_TOTAL num

This command specifies the total number of words on a horizontal video line. For NTSC video, this value should be 1716. The value must be less than 2048. If the H_TOTAL command does not appear in the pattern definition file, the number of horizontal samples per line defaults to the NTSC value of 1716.

V_TOTAL num

This command specifies the total number of video lines in the frame. For NTSC video, this value should be 525. The value must be less than 1024. If the V_TOTAL command does not appear in the pattern definition file, the number of vertical lines per frame defaults to the NTSC value of 525.

HROM_FILENAME "file name prefix"
VROM_FILENAME "file name prefix"
CROM_FILENAME "file name prefix"

These commands specify the prefixes for the names of the output files. The file name prefix must be enclosed in quotation marks. **cbgen** will append to the supplied file name prefix either "_sim" for the simulation initialization file or "_syn" for the synthesis initialization file and the appropriate file extension type. Because the file name prefix string is enclosed in quotes, space characters are acceptable in the name. If these commands do not appear in the pattern definition file, default file names of "horz_rom", "vert_rom", and "comp_rom" will be used.

HROM_INSTANCE "instance name"
VROM_INSTANCE "instance name"
CROM_INSTANCE "instance name"

These commands specify the instance names of the various ROMs. These instance names must match the instance names of the ROMs in the Verilog or VHDL code file. The instance name must be enclosed in quotation marks. If these commands do not appear in the pattern definition file, the instance names default to "HROM," "VROM," and "CROM".

HROM_INIT_STATE num

This command specifies the starting state for the HROM state machine. This is the state that the state machine will enter after being reset. The state number must be less than 2048. If this command does not appear in the pattern definition file, the HROM init state defaults to zero.

VROM_INIT_STATE num

This command specifies the starting state of the VROM state machine. This is the state that the state machine will enter after being reset. The state number must be less than 1024. If this command does not appear in the pattern definition file, the VROM init state defaults to the V_TOTAL value.

V_INCREMENT num

This command specifies the horizontal count on which the HROM will assert the inc_v signal to cause the VROM to increment to the next vertical line. The inc_v signal is actually asserted for four counts and the two least significant bits of the supplied numeric parameter are ignored. If this command does not appear in the pattern definition file, the value defaults to 1440.

Palette Blocks

The palette block defines the colors that will be used in a pattern. Either one or two palette blocks may be defined as specified with the PALETTES command described previously. Generally only one palette is used, but if the pattern generator has a pattern select input to the CROM, this pattern select bit can select between two different color palettes.

A palette block begins with a line containing the PALETTE command and the name of the palette. A palette must be given a name.

After the PALETTE command line comes a series of color definition lines. One color in the palette is defined on each separate line. The palette block ends with a line containing the END command. Anything else after the END command on the same line will be ignored, so you can use a command like END PALETTE.

Each color definition line begins with the name of the color followed by the color type as indicated by the reserved words TYPE0 and TYPE1. After the type code, an optional IS word may be used as a separator before either three or four numeric parameters are supplied to define the components of the color.

TYPE0 colors are specified with three components in the following order: Cb, Y, and Cr. The component values are specified in decimal and are 10-bit values. The single Y value is repeated for both samples of the color.

TYPE1 colors are specified with four components in the following order: Cb, Y0, Cr, and Y1. This type allows the specification of different Y values for the two samples of the color.

IMPORTANT: The first color definition line of every palette block must define a color named BLANK. This color is generated during the horizontal and vertical blanking intervals.

All color component values must be supplied as 10-bit decimal numbers in the range 0 to 1023.

Below is an example of a palette block.

```
PALETTE eg1
//  name        type        Cb       Y       Cr       Y (type 1 only)
//  -------     -----       ---     ---      ---     ---
    BLANK       TYPE0  IS   512      64      512
    gray        TYPE0  IS   512     721      512
    yellow      TYPE0  IS   176     674      543
    cyan        TYPE0  IS   589     581      176
    green       TYPE0  IS   253     534      207
    magenta     TYPE0  IS   771     251      817
    red         TYPE0  IS   435     204      848
    blue        TYPE0  IS   848     111      481
    black       TYPE0  IS   512      64      512
    i           TYPE0  IS   612     244      395
    q           TYPE0  IS   697     141      606
    white100    TYPE0  IS   512     940      512
    black-4     TYPE0  IS   512     29       512
    black+4     TYPE0  IS   512     99       512
END
```

Horizontal Regions Block

The horizontal regions block defines the different horizontal regions in a test pattern. A horizontal region must be defined for each possible place on a line that a different color may be generated. For example, in the EG 1 color bar pattern, the red bar in the top color bar pattern actually occupies three separate horizontal regions, one for each of the small black and near-black PLUGE bars below it in the bottom pattern.

The horizontal regions block begins with a command line containing the command HORIZONTAL_REGIONS and ends with the END command line. In between, each horizontal region is defined on an individual line. The horizontal regions of a test pattern are defined from left to right across the video line beginning with the first active sample of the line (count 0). All horizontal count values from zero to the value specified by the H_TOTAL command must be included in a horizontal region.

A horizontal region definition line begins with the horizontal region code to be associated with the region. The code value is specified in decimal and must be between 0 and 15 if HREGION_BITS is 4 or between 0 and 31 if HREGION_BITS is 5. The horizontal region code is used as an address into the CROM and tells it which color to generate based on which horizontal region is active. Horizontal regions may share the same code so it is possible to define more than 16 or 32 horizontal regions. However, horizontal regions that share the same codes must always share the same color in each vertical region.

Three horizontal region codes must be reserved for the EAV, BLANK, and SAV horizontal regions and these three regions must be defined in the horizontal regions block.

Because of how the horizontal code value is used in the LINE_FORMATS block, it is easier to assign the codes sequentially and to put the EAV, BLANK, and SAV codes together at the end of the code space as shown in the example below.

Following the code value on the horizontal region definition line there may be an option IS keyword. After that, the extent of the horizontal region is specified by a starting horizontal count value and an ending horizontal count value separated by the keyword TO. Horizontal count values are specified in decimal and indicate the actual horizontal count (there are two counts per video sample, one for the chroma component and one for the luma component). Horizontal regions must begin on a count value that is divisible by four and must end on a value that is one less than a value divisible by four. The first horizontal region should begin at 0 and the last horizontal region should end at one less than the H_TOTAL value.

After the horizontal region's extent, the region type must be specified. Regions may be of type ACTIVE for regions in the active video space, BLANK for regions in the horizontal blanking region, EAV or SAV for the regions where the TRS symbols are generated. The EAV and SAV regions must have extents of exactly four counts.

```
HORIZONTAL_REGIONS
// code         start    end     type
// ----         -----    ----    ------
    0       IS      0 TO  207    ACTIVE
    1       IS    208 TO  259    ACTIVE
    2       IS    260 TO  415    ACTIVE
    3       IS    416 TO  519    ACTIVE
    4       IS    520 TO  623    ACTIVE
    5       IS    624 TO  779    ACTIVE
    6       IS    780 TO  831    ACTIVE
    7       IS    832 TO 1039    ACTIVE
    8       IS   1040 TO 1107    ACTIVE
    9       IS   1108 TO 1179    ACTIVE
    10      IS   1180 TO 1247    ACTIVE
    11      IS   1248 TO 1435    ACTIVE
    12      IS   1436 TO 1439    ACTIVE
    14      IS   1440 TO 1443    EAV
    13      IS   1444 TO 1711    BLANK
    15      IS   1712 TO 1715    SAV

END HORIZONTAL_REGIONS
```

Line Formats Block

The line formats block specifies the various formats used by the video lines in the test pattern. For example, the EG 1 test pattern contains three different patterns, the top 75% color bar pattern, the middle "new chroma set" pattern, and the bottom pattern with the PLUGE signals. To implement an EG 1 color bar pattern, three different line formats would be defined, one for each pattern in the EG 1 test pattern.

If two different test patterns are being defined for the pattern generator, then the line formats block should have line formats defined for both patterns. The example below includes formats for both the EG 1 and the RP 178 test patterns in the same line formats block.

The line formats block begins with a LINE_FORMATS command line and ends with the END command line. In between, each line format is specified on an individual line format definition line. Because line format definitions can be long, it is often handy to use the line continuation character to format these lines.

A line format definition begins with a name to be given to the line format. After the name, there may be an optional IN reserved word followed by the name of the color palette to be used for this line format.

After the color palette name comes a list of colors to be used for each horizontal region on the line. A horizontal region is assigned to a color with the syntax:

```
region_code IS color_name
```

For example, the command `"0 IS gray"` assigns the color gray to the horizontal region code of 0 in this line format. All samples on the video line that fall in any horizontal region having a horizontal region code of 0 will be gray in color.

It is common for several continuous regions to have the same color. If these regions have been assigned sequential horizontal region codes, a short hand command can be used to assign them all to the same color. This syntax has the format

```
region_code TO region_code ARE color_name
```

For example, the command `"8 TO 10 ARE red"` assigns horizontal region codes 8, 9, and 10 the color red.

Every horizontal region except the EAV, SAV, and BLANK regions must be assigned to a color on each line format definition line.

```
LINE_FORMATS

//   name    palette  colors to use for each horizontal code
// --------  -------  ---------------------------------------------------
top_band   IN eg1    0 IS gray         1 TO 2 ARE yellow   3 TO 4 ARE cyan \
                     5 TO 6 ARE green  7 IS magenta        8 TO 10 ARE red \
                    11 TO 12 ARE blue

mid_band   IN eg1    0 IS blue         1 TO 2 ARE black    3 TO 4 ARE magenta\
                     5 TO 6 ARE black  7 IS cyan           8 TO 10 ARE black \
                    11 TO 12 ARE gray

bot_band   IN eg1    0 TO 1 ARE i      2 TO 3 ARE white100 4 TO 5 ARE q\
                     6 TO 7 ARE black  8 IS black-4        9 IS black  \
                    10 IS black+4     11 TO 12 ARE black

rp178_ceqx IN rp178 0 TO 11 ARE ceq  12 IS ceqx

rp178_ceq  IN rp178 0 TO 12 ARE ceq

rp178_pll  IN rp178 0 TO 12 ARE pll

END LINE_FORMATS
```

Vertical Regions Block

The vertical regions block defines the extent of each vertical region in the video frame. Different vertical regions occur where the test pattern changes from one line format to another. Vertical regions must also be defined for the vertical blanking interval. Usually, more than one vertical region must be defined in each blanking interval because the field indicator bit (F) must transition during the vertical blanking interval.

The vertical region block begins with a line containing the VERTICAL_REGIONS command and ends with the END command line. In between, each vertical region is defined on a separate vertical region definition line.

The vertical region definition line begins with a vertical region code value to be assigned to the vertical region. Multiple vertical regions can be assigned the same vertical region code as long as they have the same attributes. In order to share a code, they must be in the same field and of the same type (ACTIVE or BLANK) and the video lines in the regions must use the same line format. Sharing vertical region codes is usually easier than sharing horizontal regions codes because there are often vertical regions with identical attributes. Note in the example below how vertical region code 0 is used by both vertical blanking intervals in field 1.

After the vertical region code, there may be an optional IS keyword. This is followed by the vertical region extent definition. The extent is defined with the syntax:

```
start_line TO end_line
```

All lines in the frame from 1 to the last line must be included in a vertical region.

After the region extent definition is an optional IN reserved word followed by the field definition: FIELD0 or FIELD1. This defines whether the F bit generated by the VROM will be 0 (FIELD0) or 1 (FIELD1).

After the field definition is the vertical region type, either ACTIVE for regions in the active video region or BLANK for regions in the vertical blanking interval.

ACTIVE region definitions must end with one or two format assignments, depending on how may palettes are defined. BLANK regions do not have a format assignment. The format assignment syntax is:

```
palette_name IS line_format_name
```

This specifies that for all video lines in this vertical region, if the named palette is selected, the given line format should be generated.

```
VERTICAL_REGIONS

// code    start   end     field    type    palette format    palette format
// ----    -----   ---     -----    ----    ------- ------    ------- ------
    0   IS     1 TO   3   IN FIELD1   BLANK
    1   IS     4 TO  19   IN FIELD0   BLANK
    2   IS    20 TO  20   IN FIELD0   ACTIVE   eg1 IS top_band  rp178 IS rp178_ceqx
    3   IS    21 TO 141   IN FIELD0   ACTIVE   eg1 IS top_band  rp178 IS rp178_ceq
    4   IS   142 TO 196   IN FIELD0   ACTIVE   eg1 IS top_band  rp178 IS rp178_pll
    5   IS   197 TO 217   IN FIELD0   ACTIVE   eg1 IS mid_band  rp178 IS rp178_pll
    6   IS   218 TO 263   IN FIELD0   ACTIVE   eg1 IS bot_band  rp178 IS rp178_pll
    1   IS   264 TO 265   IN FIELD0   BLANK
    0   IS   266 TO 282   IN FIELD1   BLANK
    7   IS   283 TO 402   IN FIELD1   ACTIVE   eg1 IS top_band  rp178 IS rp178_ceq
    8   IS   403 TO 459   IN FIELD1   ACTIVE   eg1 IS top_band  rp178 IS rp178_pll
    9   IS   460 TO 480   IN FIELD1   ACTIVE   eg1 IS mid_band  rp178 IS rp178_pll
    10  IS   481 TO 525   IN FIELD1   ACTIVE   eg1 IS bot_band  rp178 IS rp178_pll

END VERTICAL_REGIONS
```

Running cbgen

After the pattern definition file has been created, the **cbgen** utility is used to generate the ROM initialization files. The **cbgen** utility is a command line utility and should be executed in a command line shell in Windows.

The syntax for executing the **cbgen** utility is:

```
cbgen [-s synth_tool] [-l language] input_filename
```

The optional –s flag specifies which synthesis tool to target with the initialization files. Different synthesis tools have slightly different syntax for initialization of Xilinx block RAMs. The choices are: XST, SYNOPSYS (for FPGA Express), LEONARDO, and SYNPLIFY. If the –s flag is not provided, the synthesis tool defaults to XST.

The optional –l flag specifies which language to use for the initialization files. The choices are: VERILOG, VHDL, and XDL. If the –l flag is not provided, the language defaults to Verilog. If the XDL option is chosen for the language, then the –s flag is ignored.

The input_filename must be the full name, including extension, of the pattern definition file.

Using the RAM Initialization Files

Verilog

The **cbgen** utility creates two Verilog files for each of three ROMs. One Verilog file contains the simulation initialization code and the other contains the synthesis initialization code. If the simulation and synthesis tools used supports the Verilog `include directive, then simply modify the six include directives in the **vidgen.v** file to include the correct file. If the tools do not support the include directive, insert the contents of the appropriate initialization file in place of each include directive in the **vidgen.v** file.

The simulation initialization code is surrounded by commands to cause the synthesis tool to ignore the simulation specific code. The synthesis specific code is written in the form of a Verilog comment block and is ignored by the simulation tool.

VHDL

The **cbgen** utility creates two VHDL files for each of the three ROMS. One VHDL file contains the simulation initialization code and the other contains the synthesis initialization code.

Unlike Verilog, VHDL does not have a file inclusion directive. So, you must use an editor to manually insert the files generated by **cbgen** into the **vidgen.vhd** file at the places indicated by the comments.

The initialization code for simulation is in the form of a generic map for each ROM. This generic map is surrounded by directives to cause the synthesis tool to ignore it.

The synthesis initialization code is a series of attribute definitions. These are user defined attributes and have to be declared before they can be used. The synthesis init file for the HROM defines all the attributes used by the initialization code. Therefore, the HROM synthesis file must be inserted in the **vidgen.vhd** file before the synthesis files for the other two ROMs.

XDL

XDL is a Xilinx utility included with ISE. XDL converts an NCD file to a text file so that it can be manually edited and then converts the text file back to an NCD file. This allows a design that has been run through the synthesis and place-and-route tools to be manually edited. Using the XDL files created by **cbgen** allows the block RAMs of the **vidgen** pattern generator to be updated with new initialization values without having to resynthesize or run PAR.

The procedure for using the XDL files is:

1. Run **cbgen** with the –l XDL flag to generate the three initialization XDL files, one for each ROM.
2. Convert the NCD file of the FPGA design to an XDL text file using the following command:

   ```
   xdl -ncd2xdl ncd_filename
   ```

   XDL will create a text file with the same name as the NCD file with a .xdl extension.
3. Open the XDL file created by the xdl utility in a text editor. Search for the HROM instance. The first few lines of the HROM instance will look like this:

```
inst "HROM" “RAMB16” , placed BMR8C1 RAMB16_X0Y0 ,
 cfg "ENAINV::ENA CLKAINV::CLKA WEAINV::WEA SSRAINV::SSRA
      CLKBINV::CLKB WEBINV::WEB ENBINV::ENB SSRBINV::SSRB
      WRITEMODEA::READ_FIRST PORTA_ATTR::1024X18 RAMB16A:HROM.A:
      WRITEMODEB::READ_FIRST PORTB_ATTR::1024X18 RAMB16B:HROM.B:
INIT_00::0X0010000F000E000D000C000B000A000900080007000600050004000300020001
INIT_01::0X0020001F001E001D001C001B001A001900180017001600150014001300120011
```

Replace all the lines starting with the line beginning with WRITEMODEA through, but not including the final line of the instance block, with the contents of the HROM XDL file. The last line of the instance block is a line with a semicolon (;) on a line by itself. Leave the last line intact.

Repeat step 3 for the VROM and CROM instances.

4. Convert the XDL file back to an NCD file using the XDL utility like this:

   ```
   xdl -xdl2ncd xdl_filename [ncd_filename]
   ```

An optional ncd_filename can be supplied, otherwise the original NCD file will be overwritten.

5. The resulting NCD file can be processed to generate a bit file that can be loaded into the FPGA.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/07/02	1.0	Initial Xilinx release.