Journal of Imaging Science and Technology® 50(1): 80–92, 2006.

A Method of Automatically Searching the Sampling Frequency and Sampling Phase for Graphic Digitizer in Pixelated Display Applications

Liming Xiu and Hugh Mair
Texas Instruments Incorporated, Dallas, Texas 75243

Abstract. *When analog red, green, blue (RGB) signals of a video/graphic image need to be displayed on pixelated display devices, graphic digitizers have to be utilized to convert the analog signals to digital signals. In such an application choosing correct sampling frequency for the digitizers is essential since an incorrect frequency will impair the recovery of images encoded in the analog signals. This article presents an algorithm that searches automatically for the sampling frequency. The phase of the sampling clock is also critical since inadequate phase control can create undesirable visible artifacts. Thus, a second algorithm for automatically searching for the appropriate sampling phase is presented in this article as well. These algorithms can be applied in pixelated display applications.* © *2006 Society for Imaging Science and Technology.* [DOI: 10.2352/J.ImagingSci.Technol.(2006)50:1(80)]

INTRODUCTION

One of the newest accessories developed for personal computers (PCs) is the liquid crystal display (LCD). Compared to its cathode ray tube (CRT) counterpart, LCD not only offers space saving, lower radiation emission, lower power consumption, but also delivers comparable video performance with a sleek, cutting-edge look. Since analog display interfaces are still dominant in the PC industry, the use of graphic digitizers to convert red, green, blue (RGB)/YUV signals from analog domain to digital domain becomes a vital technology of interfacing LCD to any analog image source. There are several devices in the market that are aimed at this application, such as device A[1] and device B.[2] These devices all contain triple digitizers, or analog to digital converters (ADC), which convert the red, green, and blue analog signals to digital signals simultaneously.

Figure 1 is a sample block diagram that shows the working principle of any pixelated display system. Graphic or video images are first generated inside the video/graphic card. Then they are converted to analog wave forms by digital to analog converters (DACs). In most cases, these analog wave forms will drive an analog CRT monitor. However, when a digital display device (with fixed pixel structure) is used to display the images, these wave forms need to be converted back to digital values by ADCs as shown in the diagram. In such an application, the key effort to successful image recovery is to determinate the sampling frequency and sampling phase (the point in time of triggering the ADC within a sampling clock's cycle) for the ADCs. These two factors have a dominant impact on the quality of the recovered images.

Phase lock loop (PLL), as shown in Fig. 2, is commonly used for generating the ADCs' sampling frequency. When the PLL is locked to the horizontal sync (HS), its output is used as a sampling clock for the ADCs. The dividing ratio of the programmable divider is typically programmed to the *number of total pixels per line* for a given video/graphic mode. Thus, the resulting sampling clock's frequency is the HSs frequency multiplied by *number of total pixels per line*. Ideally, by this mechanism, this sampling clock will have the same frequency as that of the pixel clock in the video card. However, this is not always guaranteed due to: (1) The low frequency HS usually is very noisy with significant jitter. Furthermore, its frequency accuracy is not very reliable. (2) The video card pixel clock frequency might not be exactly equal to the frequency specified in the Video Electronics Standards Association (VESA) specification;[3] a certain degree of error might exist. As a result, the original image encoded in the analog signals cannot be recovered reliably. Therefore, the process of determining the sampling frequency has to be carried out in the real application environment. In this article, an algorithm is presented to search for this sampling frequency, which can be implemented readily in real video/graphic application systems.

The analog display signals and other timing reference signals are typically coupled between the graphic source and the digital display device through separate electric paths. Due to varying lengths and impedances of cables, the reference signals and the analog display signals can be received by the display device at slightly different times. Thus, deciding when to sample the analog display signals (adjusting the clock edge within a sampling clock cycle) also has great impact on the quality of the recovered images. This exact point in time of sampling, within every sampling clock's cycle, is defined as the sampling phase. The task of determining an appropriate sampling phase could be done by a user through visual inspection. However, different users may apply different judgments when choosing "good" images. Also, such manual techniques are often cumbersome even for the knowledgeable users, and are thus unattractive. This article presents an algorithm that employs several parameters to

Received Jan. 18, 2005; accepted for publication Mar. 17, 2005.
1062-3701/2006/50(1)/80/13/$20.00.

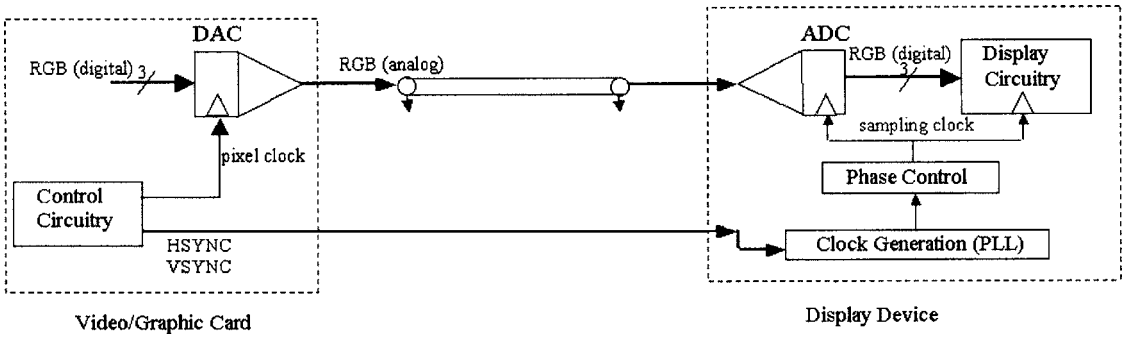

Figure 1. The pixelated display system.

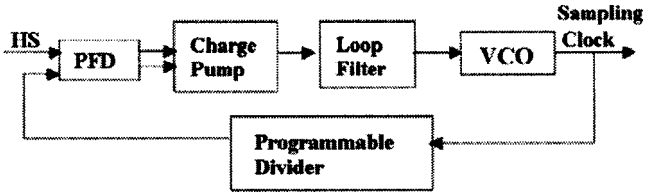

Figure 2. On-chip PLL.

measure the quality of the recovered images. These parameters provide objective criteria to differentiate good images from "bad" images.

In this article, we will address the frequency searching algorithm, the phase searching algorithm, the test report of the algorithms, and the implementation guideline for the algorithms.

THE FREQUENCY SEARCHING ALGORITHM

The Sampling Frequency

In the VESA generalized timing formula standard,[4] the principle objective is to allow predictable timing parameters to be derived from minimal information. Using this standard, it is possible to construct a complete set of timing parameters given certain basic information. One of the crucial elements in this standard is the image pixel format. For example, an image format of 800×600 symbolizes an image that has 800 active pixels in the horizontal direction and 600 active pixels in the vertical direction.

Figure 3 is the typical wave form of a video signal. Since pixels in the active video range depict the information that can be examined by viewers, any error on the *number of active pixels per line* will be immediately sensed by them visually. This number has to be honored precisely by the definition of the given image format. Thus, the correct sampling clock frequency is defined as the frequency that generates the exact *number of active pixels per line* in the active video region. This frequency *should* equal the pixel clock frequency of the video card. Since the pixel clock is not transmitted from the video card to the display device as shown in Fig. 1, our approach of finding this frequency is to adjust the dividing ratio of the PLL divider and make the pixel number in the active video region equal the *number of active pixels per line* defined in the VESA specification.

The Outline of the Algorithm

Figure 2 is the block diagram of phase lock loop used for this application. PFD is the frequency and phase detector that converts the frequency or phase difference of its two inputs to voltage signals. VCO stands for voltage control oscillator whose oscillation frequency is depended upon its input control voltage. The divider in the feedback loop is used to divide the VCO frequency down to certain value. The charge pump and loop filter are used to convert and filter the PFD output to a certain signal which can be utilized by the VCO. The output of the VCO (the ADCs' sampling clock) is locked to the HS through the divider. The dividing ratio determines the VCO frequency. Ideally, this ratio should be the *number of total pixels per video line* suggested in the VESA specification. However, it does not always produce the right pixel number in the active video region as previously addressed. The goal of this searching

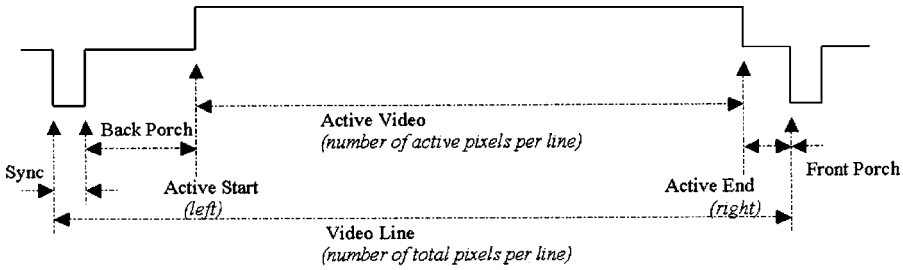

Figure 3. Wave form of typical video signal.

process is to find the correct value for this dividing ratio. This section will develop conceptual framework of this process.

Definitions

DR: *dividing ratio of the programmable divider*;
HTOT: *number of* **total** *pixels per video line suggested in the VESA specification*
HADR: *number of* **active** *pixels per video line defined in the VESA specification*; and
HADRM: **measured** *number of pixels in active video region for a video line*.

An initial starting frequency is needed for this searching process. It could be produced from the PLL by setting the dividing ratio to the HTOT for a given VESA mode. In device A there is an on-chip frequency synthesizer[5] that can generate any frequency in a certain range. The method of detecting a VESA mode is to use a known high frequency to measure the HS and VS frequencies and compare them to the numbers defined in the VESA specification.

This frequency searching algorithm can be outlined in the following steps:

1. Set DR to an initial value, usually HTOT
2. Compute HADRM
3. If HADRM is not equal to HADR
   then reset DR by using:

$$\mathrm{DR(new)} = (\mathrm{HADR}/\mathrm{HADRM}) * \mathrm{DR(old)} \qquad (1)$$

4. Compute HADRM again.
   If HADRM is not equal to HADR
   then repeat step 3.
   Done when HADRM is equal to HADR.

The earlier steps should be executed on all three primary colors. The largest HADRM among red, green and blue is the final HADRM. This procedure can be appreciated by an example of XGA (1024×768) at 60 Hz refresh rate (HADR = 1024).[3] If the DR is initially set to 1296 arbitrarily, the computed HADRM of this case is 987, which is not correct for this image format. The DR(new) then can be calculated as: $(1024/987)*1296=1344$. This number (1344) will be stored in the PLLs programmable divider and the resulting sampling frequency will guarantee 1024 pixels in the active video region.

Since the searching process has to sample multiple image frames, it is required that the image content does not change during this time so that all calculations are based on same information.

Detailed Description of the Algorithm

The key task of this frequency searching algorithm is to find the number of active pixels in active video region, or HADRM. Referring to Fig. 3, it is clear that this task is equivalent to the work of finding the left and right edges of active video. Subsequently it can be divided into three subtasks: (1) Find blank level of RGB signals. (2) Find maximum value of RGB signals. (3) Find left and right edges.

Find blank level

For the first video line in an image frame, do the following averaging calculation.

$$\mathrm{blank} = \sum_{N} \mathrm{value}/N, \qquad (2)$$

where value is sampled, or ADC converted, red, green, or blue analog signal of a pixel; N is the number of total pixels in this video line. This calculation is valid since the first line of any image frame is always blank in all VESA modes.

Find maximum RGB value

For an n-bit ADC, the output value is in the range of $[0\text{-}(2^n-1)]$. The digital RGB value associated with any pixel must fall in this range. The maximum RGB value is defined as the maximum among those values generated from all the pixels in this frame. This parameter, max_val, can be found straightforwardly by a simple routine.

Find left and right edges

A threshold is firstly defined as

$$\mathrm{threshold} = \mathrm{factor} * (\mathrm{max\_val} - \mathrm{blank}). \qquad (3)$$

Factor is a predetermined fractional number between 0 and 1. As can be readily appreciated from Fig. 3, the RGB values of pixels in the active video region are significantly different than those in the so-called porch regions where RGB values are at a blank level. This threshold is a value between max_val and blank and is used to help to identify the start and the end of the active video region. In our test experience, 0.25 is a good choice for this parameter. Second, starting from the beginning of a current video line, if we find a series of consecutive pixels whose RGB values are all greater than threshold, then the first pixel of this series is the left edge of the active video. Third, starting from some point in the middle of a current video line to the end of same line, if we find a series of consecutive pixels whose RGB values are all smaller than threshold, then the first pixel of this series is the right edge of the active video.

From Fig. 3, it is clear that for each video line there is a Back Porch at the blank level before the active video, and a Front Porch after the active video. The above procedure will give us the locations of the start/end of the video signal whose RGB value is significantly different than that of the blank level. "A series of consecutive pixels" is used since we want to eliminate the random error of one pixel. In our experience, five pixels are enough to qualify for a series of consecutive pixels.

For all video lines in a frame, it is required to find the left and right edges of each individual line. At the end of this frame the leftmost edge (smallest X location of *Active Start*) among all the lines in this frame is this frame's left edge, the rightmost edge (largest X location of *Active End*) among all the lines in this frame is this frame's right edge. The above calculations must be performed for all three colors. The leftmost edge among all the colors is the final left edge. The rightmost edge among all the colors is the final right edge.

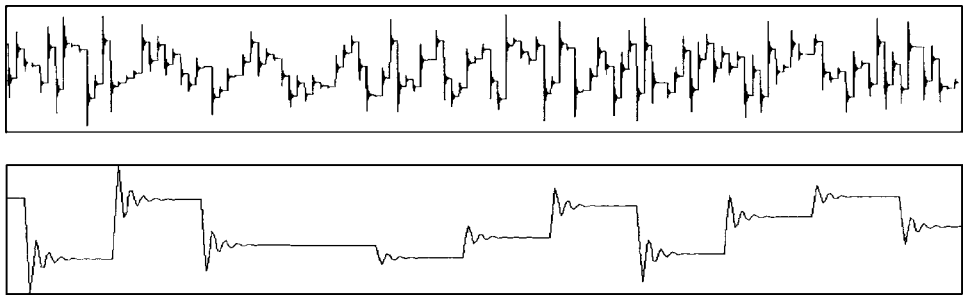

Figure 4. The ideal wave form at the ADCs input.

Finally, HADRM is calculated as

$$\text{HADRM} = \text{right\_edge} - \text{left\_edge}. \quad (4)$$

The Condition for this Algorithm to Fail

(1) The RGB value of the left or right edge of the **entire** frame is at the blank level or very close to the blank level. To be precise, the values of the first or last few **active** pixels of **all** video line in the frame are all smaller than the threshold.
(2) **And** condition 1 is true for **all** the three colors.

For window-based PC applications, most users use some kind of background. The only scenario that will fail this algorithm is when no background is used, or applications are running on a totally **black** background.

Related Works

In Ref. 6 a method of searching sampling frequency by using certain predetermined test patterns is presented. First, some sequences of test patterns are encoded in an analog video source and transmitted to a digital display device where the analog signal is converted to sequences of the sampled values. Then, the digital display device determines whether the sampled values equal one of the sequences of the test patterns based on the predetermined convention. The digital display device changes the sampling frequency until the sampled values equal one of the sequences of the test patterns. Finally, the corresponding frequency will be used as ADCs' sampling frequency when the match is found. Comparing this sequence of operations to our method: (1) It needs test patterns encoded in analog video source, which requires additional hardware/software. Unfortunately the display device engineers do not usually have control on how the video source is configured/designed. (2) The operation is a feedback system. But this feedback scheme does not specify how the next sampling frequency should be determined (just vary the sampling frequency). This poses a convergence problem. In our method, we have a mechanism to determine the next sampling frequency.

In Ref. 7, the method of choosing sampling frequency is to detect the VS and HS frequencies and compare them to the commonly used standard video time data specified in the VESA guideline.[3] The VESA mode whose time data most closely resemble the detected VS and HS is the desired VESA mode. The corresponding pixel frequency will be used as the sampling frequency. The problem of this scheme is that the pixel frequency specified in VESA documents is just a guideline. In real applications deviations do exist, and certain degree of frequency error on the pixel clock is unavoidable. Actually, the sampling frequency obtained by this method is the initial starting frequency in our method as mentioned in the outline of the algorithm.

THE PHASE SEARCHING ALGORITHM

The Sampling Phase

As mentioned in the Introduction, the graphic digitizers, or ADCs, are fed by the DACs that are located in PC video/graphic card. The DACs' output is stepwise wave forms with overshoot or undershoot at the beginning or the end of pixel boundary if adjacent pixels have different RGB levels. As shown in Fig. 4, the top wave form belongs to a series of pixels of one video line, of one particular color. The bottom one is the zoom-in on one section of the top wave form. The step size in the X axis is the size of one pixel in the time domain. The Y axis represents the RGB value.

When the ADCs sampling frequency is correctly determined one voltage level within each pixel will be sampled by the ADC and be converted to a digital value. This operation is executed at each clock cycle, pixel by pixel sequentially. When to sample the analog signal, within a pixel boundary, has great impact on the converted digital value. Searching for an appropriate sampling phase is to find the best "point in time" to trigger the ADC and consequently generate the best image possible. Both device A (Ref. 1) and device B (Ref. 2) have 32 steps within each clock cycle. These steps, which correspond to the sampling phases, can be used to trigger the ADC at specific time.

In the past some studies have been done on the sampling clock's jitter impact to ADC conversion,[8] and to digital communication.[9] Research on the reconstruction of the original image from several phase-shifted images is presented in Ref. 10. The challenge in our application is different. We need to find the best sampling point so that the resulting image most closely resembles the original image. As can be appreciated from Fig. 4 it is clear that good sampling points are in the area where the wave form is flat. This is the area where the signal finally stabilizes. The overshoot/

undershoot area is where the signal is still in transition and should be avoided for sampling. In reality the "flat" areas are not as well developed as shown in Fig. 4. The algorithm has to define certain quality critera and use them to make the best choice among several candidates.

The Analog RGB Wave Form Reconstruction

This phase searching algorithm is based on the RGB wave form reconstructed at the ADCs output. Oversampling the original analog signal with a higher speed clock is one way of achieving this reconstruction. However, it requires a much higher frequency clock and high speed ADCs. An alternative is to sample the same signal multiple times, each time with a different phase. Then the RGB wave form can be reconstructed by the data collected at these phases. The clock phase movement has to be monotonic when the phase control is swept from one end to the other end. This is true for both devices A and B, according to their datasheets. Also as in the case of the frequency searching algorithm, the image content cannot change during the search process. The procedure for the reconstruction of analog RGB wave form can be described as follows:

(1) Select one video line according to certain criterion.
(2) Sample this video line multiple times, each time with a unique sampling phase.
(3) Build the wave form by a two-dimension array: wf[pixel][phase].

For advanced VESA modes the pixel clock frequencies are well above 100 MHz. It is understandable that the ADC performance will inevitably degrade with increasing operating speed. To improve data quality certain filter functions can be applied to wf[pixel][phase]. Low-pass moving average FIR filter is good candidate for this application. The following are formulas for 3-tap, 5-tap, and 7-tap moving average filters:

$$\text{3-tap-value}[k] = (\text{Value}[k-1] + \text{Value}[k] + \text{Value}[k+1])/3, \quad (5)$$

$$\text{5-tap-value}[k] = (\text{Value}[k-2] + \text{Value}[k-1] + \text{Value}[k] + \text{Value}[k+1] + \text{Value}[k+2])/5, \quad (6)$$

$$\text{7-tap-value}[k] = (\text{Value}[k-3] + \text{Value}[k-2] + \text{Value}[k-1] + \text{Value}[k] + \text{Value}[k+1] + \text{Value}[k+2] + \text{Value}[k+3])/7. \quad (7)$$

Parameters to Measure the Quality of Recovered Image

Laboratory experiments show that when sampling at the "flat area" of each pixel, the recovered image has better quality than those captured at other sampling points. Therefore, searching for the best sampling phase is equivalent to the identification of the "flattest" point in each pixel's wave form section. Several parameters will be created below to measure the "flatness" of each data point of this reconstructed wave form.

First derivative

$$\text{fd}[\text{pixel}][\text{phase}(n)] = \text{abs}(\text{wf}[\text{pixel}][\text{phase}(n+1)] - \text{wf}[\text{pixel}][\text{phase}(n)]) + \text{abs}(\text{wf}[\text{pixel}][\text{phase}(n)] - \text{wf}[\text{pixel}][\text{phase}(n-1)]). \quad (8)$$

The *first-derivative* used here is not exactly the same as the term defined in the Calculus. Instead of two data points, it uses three points to calculate the *first-derivative* of the middle point. Furthermore, absolute values are used in the calculation to ensure that its magnitude corresponds to the flatness of the current point. Since the wave form of **one** pixel is composed of multiple (in our case 32) data points this fd[pixel][phase] is one way of measuring the wave form flatness at each data point, or sampling phase.

Second derivative

sd[pixel][phase] is obtained by applying Eq. (8) to fd[pixel][phase]. This parameter can measure the second order flatness. For both fd[pixel][phase] and sd[pixel][phase], the phase that minimizes this function is the point where the wave form is at its flattest. Consequently, it is the desired sampling phase.

Distance

The *distance* is defined as

$$\text{dist}[\text{pixel}][\text{phase}] = \text{abs}(\text{wf}[\text{pixel}][\text{phase}] - \text{ref}), \quad (9)$$

where ref is a reference and can be one of the following:

(a) average value of a pixel

$$\text{ref}[\text{pixel}] = \sum_{M} \text{wf}[\text{pixel}][\text{phase}]/M, \quad (10)$$

where M is the number of sampling phases available within a pixel.
(b) ref[pixel][phase] is wf[pixel][phase] passed through the 3-tap filter.
(c) ref[pixel][phase] is wf[pixel][phase] passed through the 5-tap filter.
(d) ref[pixel][phase] is wf[pixel][phase] passed through the 7-tap filter.

In case (a), the ref is a variable which represents the "true" value of a **pixel** since it is the average RGB value of all the available points (sampling phases) within this pixel. In cases (b), (c), and (d), the ref is a variable that serves as the true value of a **sampling point**. The *distance* is used to measure the deviation between the sampled value and the true value. The smaller the dist[pixel][phase] is, the better the quality of this sampling phase is. This statement is based

on the observation that the possibility of the signal having a transition at this point is slim when this deviation is small. This might not be true for every pixel. But for the statistics of many pixels, this is generally true. Using *distance* is just another way of quantifying the quality of sampling phases, intuitively.

The Most Active Line and High Quality Pixels

In parameters to measure the quality of recovered image the parameters *first-derivative*, *second-derivative*, and *distance* are defined on the data of **one** pixel (with a pixel index). For the benefit of reducing random error, more pixels are preferred. Ideally, all pixels in a frame should be used to build arrays wf[pixel][phase], fd[pixel][phase], sd[pixel][phase], or dist[pixel][phase]. But this requires a huge amount of memory. One way of reducing the memory requirement is to use just one line of pixels, such as the "most active" video line described below. An alternative is to find a series of "high quality" pixels so that the memory usage can be further reduced.

In an image frame, usually there are some areas of "high activity" and some areas of "low activity." In high activity areas, colors vary dramatically. In other words, the RGB values of the pixels in these areas are significantly different than each other's. These high activity areas are most sensitive to sampling phase. Consequently, these pixels contain more phase information. These special pixels are denoted as most active video line. They will be used to build the array wf[pixel][phase] and the likes.

Searching for the most active video line

Several parameters can be used as references when comparing characteristics of different video lines.

Total RGB statistic of a video line, or E_{rgb}, can be defined as

$$E_{\mathrm{rgb}} = \sum_{N} (rv + gv + bv), \tag{11}$$

where N is the number of pixels in this video line, rv, gv, and bv are red, green, and blue sampled RGB values.

Red (or Green, Blue) switch statistic of a video line, or SE_r, is

$$SE_r = \sum_{N} \mathrm{abs}(rvc - rvp), \tag{12}$$

where rvc, rvp represent the red RGB values of **current** pixel and **previous** pixel, respectively.

RGB switch statictic of a video line

$$SE_{\mathrm{rgb}} = SE_r + SE_g + SE_b. \tag{13}$$

The most active line can be identified as the line with maximum SE_{rgb}. The other parameters (E_{rgb}, SE_r, SE_g, SE_b) can be used to quantify the confidence level of this found most active line.

Searching for the high quality pixels

These pixels can be identified by using the following critera:

(a) These pixels must be spatially consecutive.
(b) The red (or green, blue) values of adjacent pixels must be different.
(c) The value differences of adjacent pixels must be greater than a predetermined low threshold.
(d) The value differences of adjacent pixels must be smaller than a predetermined high threshold.

The low threshold is needed because a portion of wave form that contains significant overshoot/undershoot is desired. Phase information is better expressed in these types of wave forms. High threshold is required for the reason of signal integrity. If the values of adjacent pixels change too rapidly the ADC may not be able to respond properly, and the resulting wave form is not of high integrity. Our suggestion is to use 80% of the ADCs full range as high threshold and 30% as low threshold. The search for this series of high quality pixels can be performed continuously for a frame of data. At the end of the frame the longest series that satisfies the earlier criterions is the series found.

The Phase Searching Algorithm

(1) Use the sampling frequency found by the method of previous section, sample one frame of RGB signal to find the most active video line or a series of high quality pixels.
(2) Sample this most active video line or the line that contains the high quality pixels multiple times, each time with a unique phase, for all the possible phases. Then build the array wf[pixel][phase].
(3) Preprocess raw data by sending wf[pixel][phase] through a low pass filter (3-tap, 5-tap, or 7-tap moving average filter).
(4) Calculate the parameter array fd[pixel][phase], or sd[pixel][phase], or dist[pixel][phase].
(5) Sum the parameter array of step 4 **at each phase** as shown in Eq. (14), where N represents all the pixels in this most active video line or in this series of high quality pixels

$$\mathrm{fd[phase]} = \sum_{N} \mathrm{fd[pixel][phase]}. \tag{14}$$

(6) Find the minimum of array fd[phase]. The phase that produces this minimum is the desired sampling phase.

In Eq. (14), this array fd[pixel][phase] can be replaced by sd[pixel][phase] or dist[pixel][phase]. In this algorithm, the raw data have been preprocessed by moving average filters of Eqs. (5)–(7). Those filters are **mean** filters whose main advantage is simplicity. They can be implemented cheaply in hardware or software. But the high frequency components are not very well preserved. **Median** filters, which are better in preserving edges, can potentially do a better job of maintaining the phase information. However, this type of filter is expensive owing to the numerical sorting challenge in its mechanism. Savitsky-Golay filters can also be used to replace the filters in Eqs. (5)–(7). This type of filter

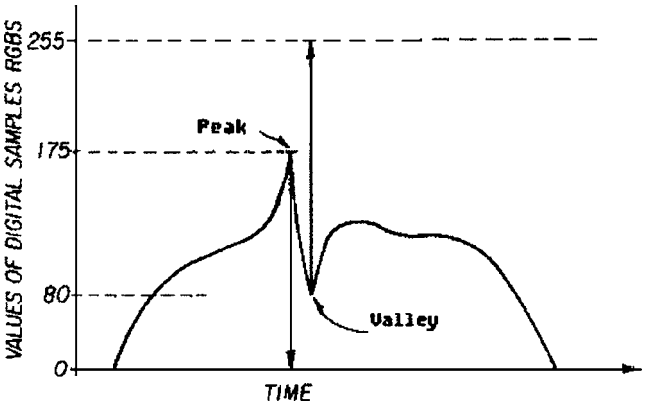

Figure 5. Peaks and valleys (see Ref. 11).

tends to preserve the high frequency components, which are needed in judging phase, better than the moving average filters, but they are also expensive.

Condition for this Algorithm to Fail

The condition for this algorithm to fail is that the series of high quality pixels or the most active video line cannot be found in a frame. In other words, this entire image frame does not contain any significant color change, or there is no useful information for people to view. A totally black or blue screen is an example which fails this phase searching algorithm. The easy solution is to switch to another more meaningful image.

Related Works

In Ref. 7 a parameter of value difference is defined as VF[pixel]=abs(*vc*-*vp*), where *vc*, *vp* are the RGB values of current and previous pixel, respectively. The phase searching method of this work is to find the pixel, max_pixel, in a frame such that VF[max_pixel] is at maximum of this defined value difference. Then, the sampling phase is varied and each phase generates a corresponding VF[max_pixel][phase]. The phase which makes this VF[max_pixel][phase] achieve its maximum is the optimum sampling phase. This approach is based on the assumption that if two adjacent pixels have different RGB values, then among all available phases the optimum phase should maximize their RGB value difference. In our opinion this method has three problems: (1) Only two pixels are used for the calculation, which poses a great possibility of random error. (2) Usually there are certain amounts of overshoot/undershoot if two adjacent pixels have significantly different RGB values, and this will make the method of VF[max_pixel][phase] invalid, or less accurate. (3) It does not utilize relationship information among different phases. In our algorithm, the fd[pixel][phase], (etc.) is built on data of consecutive phases.

In Ref. 11, values of peak and valley are defined as shown in Fig. 5. In this drawing, the *x* axis is a sequence of pixels, or the progression of time; the *y* axis is each pixel's RGB value. Those peaks and valleys are found by using pixels of a plurality of video lines, at a certain sampling phase. Then a statistical value is generated by summing up the magnitude values of these peaks and valleys. The phase that maximizes this value is the optimum sampling phase. Compared to the method presented in Ref. 7, this one is considerably better, since more pixels are used for making the decision. But this method still does not utilize the interphase relationship information for calculation. To be precise, Fig. 5 is the wave form of a series of pixels. But at each pixel position there is only one value, which corresponds to one sampling phase. In Fig. 6 of this article, each pixel position has 32 data points. In other words, instead of using the data of one sampling phase, each pixel's wave form is composed by data collected from 32 sampling phases. Since our algorithm studies the interphase relationship among all sampling phases before making the decision, it tends to generate a better result. However, it requires much more memory, and such overhead might be a disadvantage in some situations.

Reference 12 presents a method of searching for the optimum phase by detecting pixel boundaries, or transition points, in the analog wave form. Usually the sharp transition points, which are required by this method, are accompanied by significant overshoot/undershoot and oscillation ringing, which will complicate this detecting process. Compared to the methods in Refs. 7 and 11, this scheme does utilize the interphase relationship information. Compared to our method, this scheme studies the interphase relationship in real time (in the same frame) which is better than our method. However, it requires dedicated hardware (ADCs, etc.) for each sampling phase, which could be very expensive if there are many sampling phases.

TEST OF THE ALGORITHMS

The algorithms were first tested on data generated from a mathematical model during the development phase. After the algorithms were established they have been tested on devices A and B intensively.

Test Setup

As shown in Fig. 7, an image source provides HS and VS to the chip under test, or graphic digitizer (devices A or B). It also provides analog RGB signals to the digitizer. A logic analyzer is used to capture the VS, HS, and the ADC converted digital RGB data from the digitizer. The I2C bus is used to control the chip under test that is an I2C slave. The image source is a Dell personal computer. During our tests, various applications were run (MS Word™, photoeditor, game, etc.) on the PC. The output of this PC video card, which drives a regular analog monitor as well, is fed to our digitizer. A pattern generator was also used as an image source for special study, due to its excellent capability of generating various VESA signals. Since the Microsoft Windows Operating System is available in this logic analyzer,[13,14] the algorithms are coded in C/C++ and executed on the logic analyzer. The logic analyzer also functions as I2C master that controls the digitizer (such as to set the frequency register, phase register, etc.). Tests have been performed on more than ten VESA modes. The test reports of four common VESA modes, SXGA at 60 Hz refresh rate, XGA at

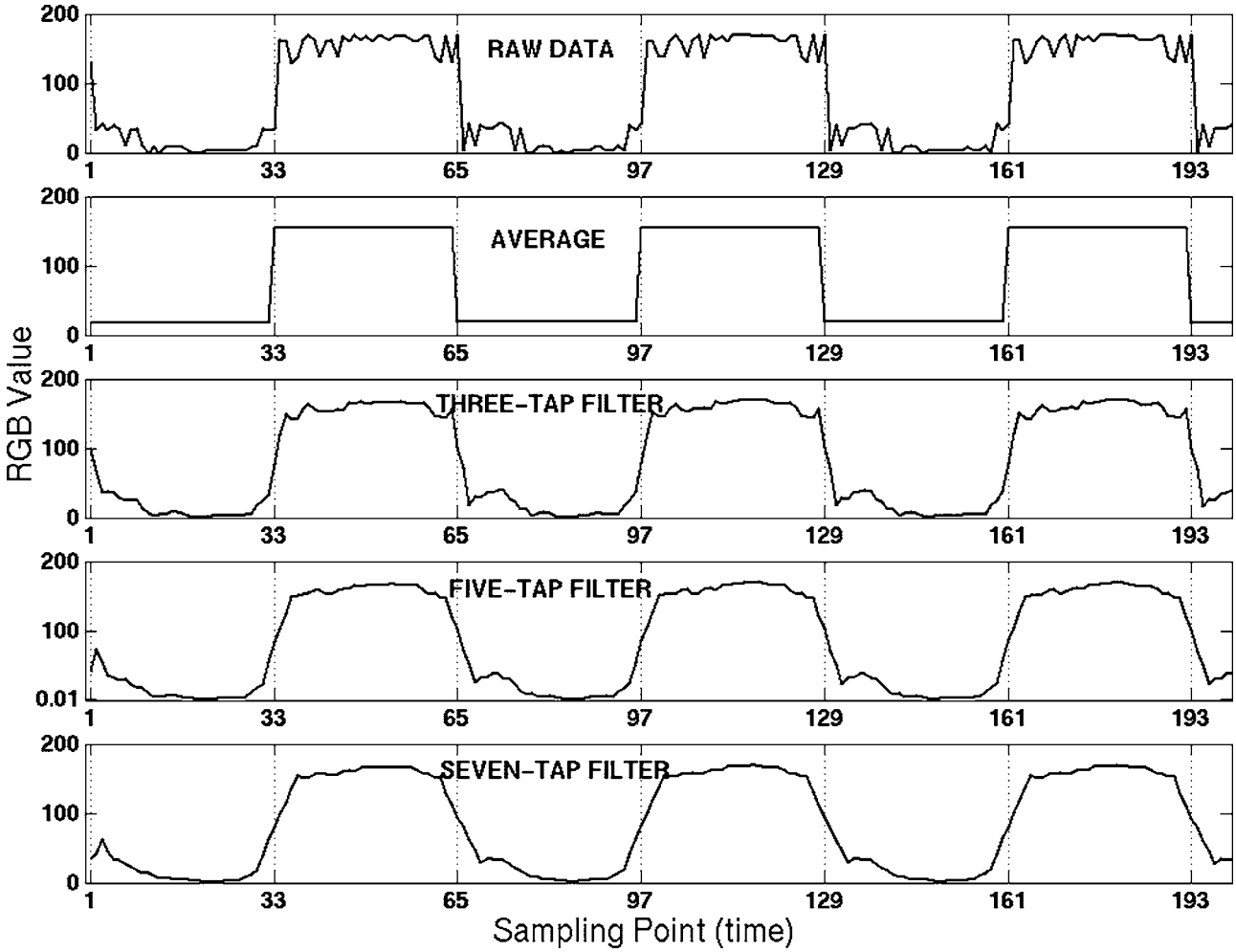

Figure 6. Raw data and filtered data.

60 Hz, SVGA at 75 Hz, and VGA at 75 Hz, are included in the next two sections.

Test of the Frequency-Searching Algorithm

As addressed in the first two sections the goal of the search for the ADCs sampling frequency is to find the correct dividing value for the PLLs feedback divider, so that the resulting number of pixels in the active video region (HADRM) equals the HADR. Tables I–IV contain the test results for XGA, SXGA, SVGA, and VGA modes. For each VESA mode, three different images are provided to the digitizer for testing. They show that the algorithm is able to find the correct dividing value when it starts with an initial value, DR(old). Furthermore, the tables also demonstrate that the feedback mechanism can derive the correct value, DR(new), by one iteration when initial error is as large as about ±10%. By using this DR(new) as the dividing ratio in the PLLs divider, the resulting images all have the correct number of pixels in the active video region as shown in the last column of each table. Also worth mentioning is the fact that in some cases the final DR(new) is not exactly equal to the HTOT which is the suggested value in VESA standard. The cause of this mismatch is a minor frequency error in the pixel clock and/or the frequency error of HS (compared to the VESA specification) in the video cards. By directly using the HTOT as the dividing ratio, without the real-time frequency searching, imperfect images could result.

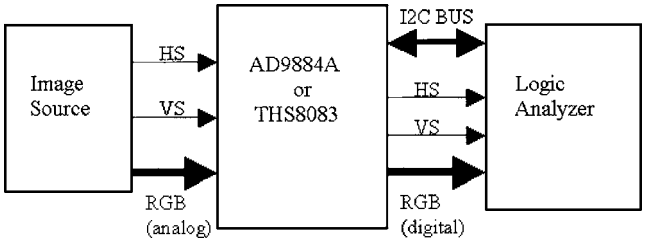

Figure 7. Test setup.

Test of the Phase Searching Algorithm

The sampling phase of the digitizer is controlled by a phase register with 32 settings. The verification approaches of the phase searching algorithm are

(1) Check to see if the desired phase found is the flat-

Table I. XGA (1024 × 768) at 60 Hz refresh rate. HTOT = 1344, HADR = 1024.

DR (old)	Error[a]	Left	Right	HADRM(old)	DR (new)	HADRM(new)
1328	−1.2%	158	1170	1012	1344	1024
1312	−2.4%	156	1156	1000	1344	1024
1296	−3.6%	154	1141	987	1345	1024
1280	−4.8%	152	1127	975	1344	1024
1264	−6.0%	150	1113	963	1344	1024
1248	−7.1%	148	1099	951	1344	1024
1234	−8.2%	147	1087	940	1344	1024
1360	+1.2%	161	1198	1037	1343	1024
1376	+2.4%	163	1212	1047	1343	1024
1392	+3.6%	165	1226	1061	1344	1024
1408	+4.8%	167	1240	1073	1344	1024
1424	+6.0%	169	1254	1085	1344	1024
1440	+7.1%	171	1268	1097	1344	1024
1456	+8.3%	173	1282	1109	1344	1024

[a]Error = (DR − HTOT) / HTOT.

Table II. SXGA (1280 × 1024) at 60 Hz refresh rate. HTOT = 1688, HADR = 1280.

DR (old)	Error	Left	Right	HADRM(old)	DR (new)	HADRM(new)
1665	−1.4%	353	1616	1263	1688	1280
1640	−2.8%	348	1592	1244	1688	1280
1617	−4.2%	343	1569	1226	1688	1280
1592	−5.7%	338	1545	1207	1688	1280
1569	−7.0%	333	1523	1190	1688	1280
1544	−7.1%	328	1499	1171	1688	1280
1521	−8.5%	323	1476	1153	1689	1280
1713	+1.5%	362	1661	1299	1688	1280
1736	+2.8%	367	1684	1317	1687	1280
1761	+4.3%	372	1708	1336	1687	1280
1784	+5.7%	377	1730	1353	1688	1280
1809	+7.2%	382	1755	1373	1687	1280
1832	+8.5%	387	1777	1390	1687	1280
1857	+10.0%	391	1799	1408	1688	1280

Table III. SVGA (800 × 600) at 75 Hz refresh rate. HTOT = 1056, HADR = 800.

DR (old)	Error	Left	Right	HADRM(old)	DR (new)	HADRM(new)
1041	−1.4%	237	1025	788	1057	800
1009	−4.5%	229	994	765	1055	800
993	−6.0%	226	978	752	1056	800
977	−7.5%	222	963	741	1055	800
961	−9.0%	218	947	729	1055	800
945	−10.5%	215	931	716	1056	800
1072	+1.5%	244	1056	812	1056	800
1088	+1.5%	247	1072	825	1055	800
1104	+3.0%	251	1088	837	1055	800
1120	+6.0%	255	1103	848	1057	800
1136	+7.6%	258	1119	861	1056	800
1152	+9.1%	262	1135	873	1056	800

Table IV. VGA (640 × 480) at 75 Hz refresh rate. HTOT = 840, HADR = 640.

DR (old)	Error	Left	Right	HADRM(old)	DR (new)	HADRM(new)
833	−0.8%	120	755	635	840	640
824	−1.9%	118	747	629	839	640
808	−3.8%	116	732	616	840	640
792	−5.7%	114	718	604	839	640
776	−7.6%	111	703	592	839	640
760	−9.5%	109	689	580	839	640
856	+1.9%	123	775	652	840	640
872	+3.8%	125	790	665	839	640
888	+5.7%	127	804	677	840	640
904	+7.6%	130	819	689	840	640
920	+9.5%	132	833	701	840	640
936	+11.4%	134	848	714	839	640
952	+13.3%	137	862	725	840	640

Table V. The measurement types.

Test type	First derivative	Second derivative	Distance
wf[pixel][phase]	Meas1	Meas5	Meas9
3-tap[pixel][phase]	Meas2	Meas6	Meas10
5-tap[pixel][phase]	Meas3	Meas7	Meas11
7-tap[pixel][phase]	Meas4	Meas8	Meas12

test point in the reconstructed RGB wave form.

(2) Check to see if fd[phase](or sd[phase], dist[phase]) has good correlation with images captured at each phase.

Since the ADCs in our digitizers are eight-bit devices, the RGB values are in the range of 0–255. The test program first searches for the high quality pixels by using a low threshold of 77 and a high threshold of 204. Then it samples this series of high quality pixels 32 times. After that, it calculates fd[phase] (or sd[phase] and dist[phase]). It also generates an image of about 30 video lines for each phase and combines all the images obtained at each phase into one big image for easy comparison. The resulting big image can be viewed by us in order to study visually the correlation between the images and the arrays fd[phase], sd[phase] and dist[phase]. Table V is the list of all possible tests that could be performed on **one** image. In our test four VESA modes (SXGA at 60 Hz refresh rate, XGA at 60 Hz, SVGA at 75 Hz, and VGA at 75 Hz), each with three different images, have been studied. In each of these 12 cases all 12 measurements have been carried out.

Figure 6 shows raw and filtered data of a series of six pixels from an image of VGA mode. Within each pixel there are 32 data points. It can be seen that the flat area is in the range of phase 19 to phase 23. (In Fig. 6, it is in the area after the middle point of each pixel.) This observation will be confirmed later by *first derivative*, *second derivative* and *distance* in Figs. 8–10.

Figure 8 shows four curves of fd[phase], which correspond to meas1, meas2, meas3, and meas4 in Table V, respectively. They are generated from a series of high quality pixels (50 pixels in this case). The *X* axis is the 32 sampling phases. The *Y* axis is the value of fd[phase], whose absolute value is not important. We are only interested in where its minimum occurs. These curves show that the minimum occurs around phase 21, which agrees with our observation from Fig. 6. They also suggest that at the two ends of sampling phases the values are greater. Hence, they should be avoided from being used to sample data.

Figure 9 and 10 are the curves of sd[phase] and dist[phase]. All the measurements in these three Figs. 8–10, except measurement 9, show that in the region of phase 19 to phase 23 the parameters reach their minima. In Fig. 11(a), there are 16 images associated with the first 16 sampling phases, phase 1 to phase 16. Each one is taken by a unique sampling phase and is composed of only 30 video lines due to space limitation. There are two lines of characters in each image. Horizontally, those images are cropped from full screen width. Figure 11(b) is the images of phase 17 to phase 32. By studying the characters' shapes and sizes, it appears to us that phase 20, 21, and 22 recover the original image best. In general, the qualities of images from phase 17 to phase 23 are better than that of others. With no surprise, these phases also have lower parameter values as shown in Figs. 8–10.

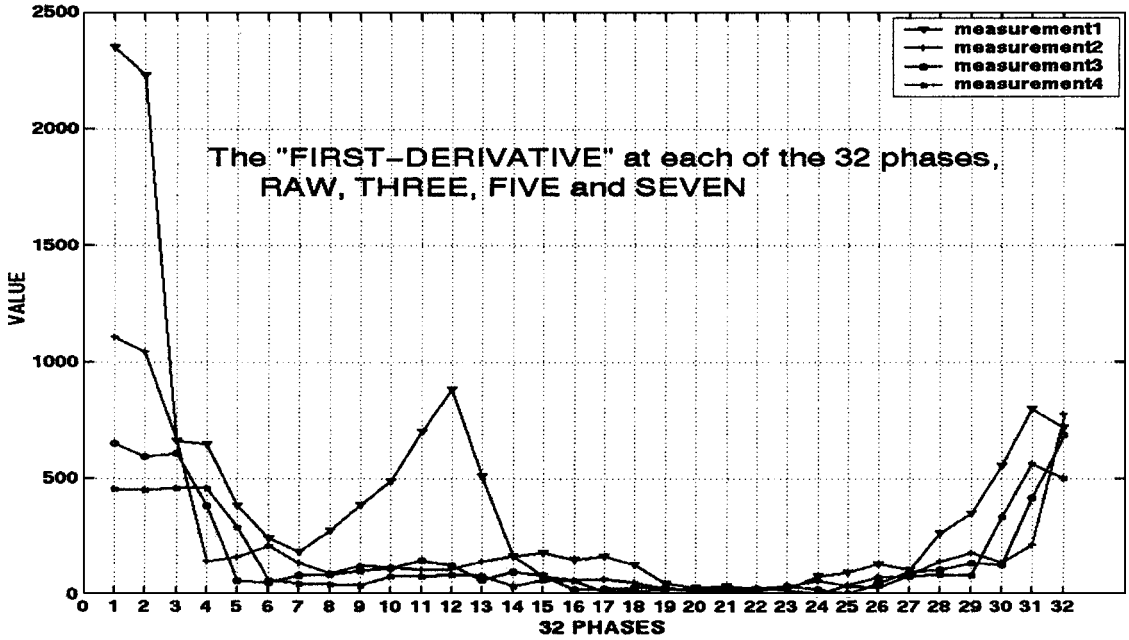

Figure 8. Four curves of the fd[phase].

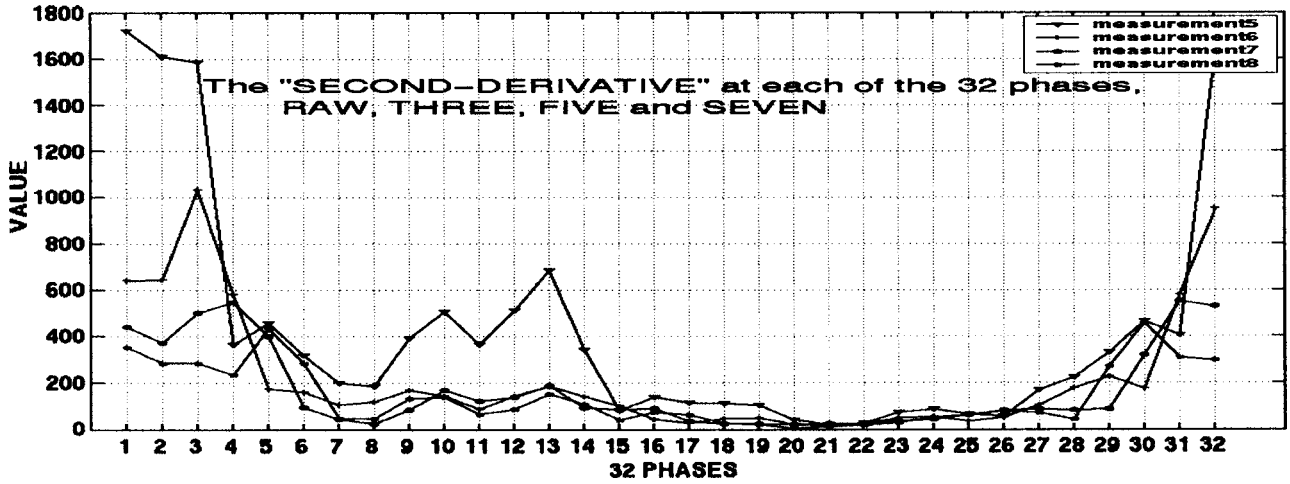

Figure 9. Four curves of the sd[phase].

In Fig. 10 when *distance* is obtained from average value of a pixel [Eq. (10), case a, meas 9], its curve does not align with other curves desirably. The reason is that the pixel's average value is not a very good reference when calculating *distance*. The high frequency information is totally lost.

IMPLEMENTATION GUIDELINE

These algorithms can be implemented in any application that has a microcontroller/microprocessor in the system. The goal of this section is to provide a guideline to help designers implement the algorithms in their system.

Partition between Hardware and Software

Since the algorithms require both a real-time data collection and a large amount of arithmetic calculation, a well defined partition between hardware and software can ensure decent system performance with a low resource cost. The partition can be discussed in following two scenarios.

A frame buffer is available

If the algorithms are implemented in a system that has a frame buffer then the tasks of finding blank levels, maximum values, and active video edges can all be accomplished by using the data stored in this frame buffer. The high quality pixels or most active video line can also be found by using these data. The task of collecting data for phase searching requires sampling a video line multiple times, which can be done by utilizing this frame buffer as well. Therefore, the algorithms can be implemented in software plus this frame buffer. No extra hardware is needed,

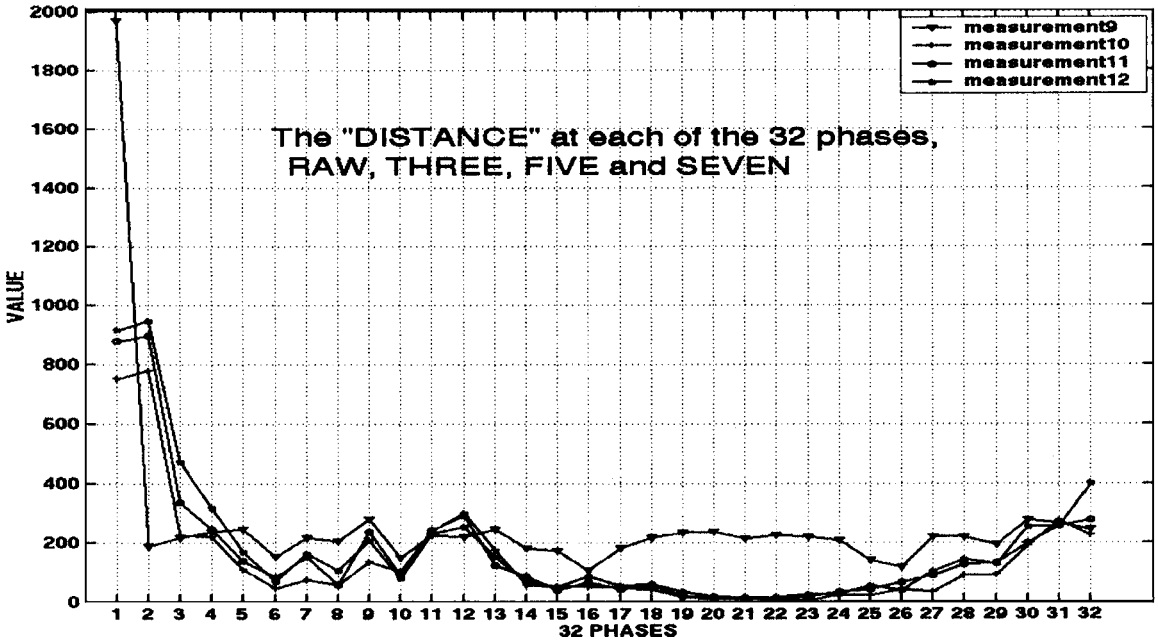

Figure 10. Four curves of the dist[phase].

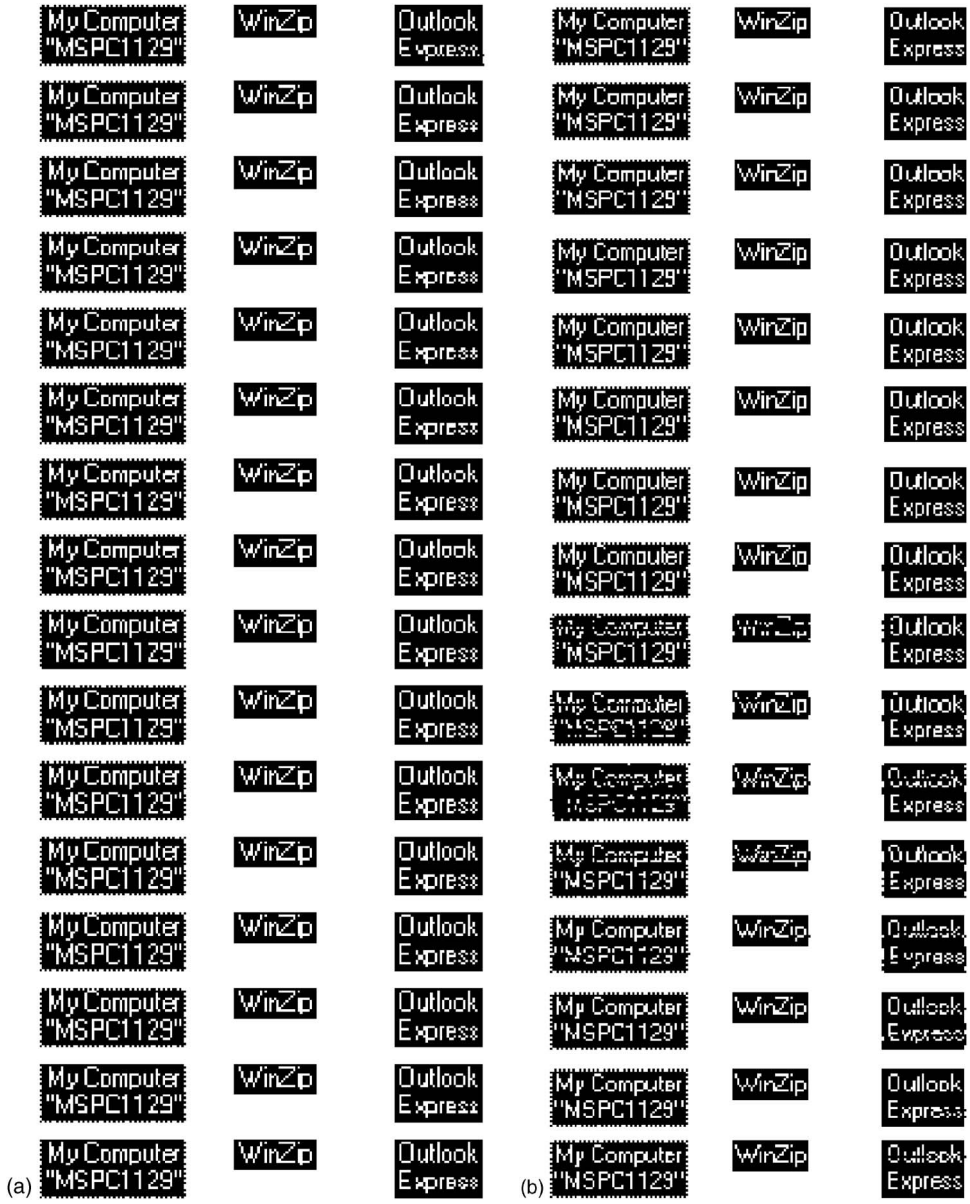

Figure 11. The combined image of (a) the first 16 phases and (b), the second 16 phases.

except the microcontroller that can be shared with other functions.

Without frame buffer

By examining the procedures described earlier, it is logical to partition all the tasks into three hardware blocks:

BLK1: *find blank levels and maximum RGB values of an image frame*;

BLK2: *find left and right edges of active video, search high-quality pixels*;

BLK3: *collect data for phase calculation*.

The task of finding blank levels and maximum values needs real-time RGB data. This can only be accomplished in hardware. Both blank levels and maximum values will be available at the end of the first frame. Finding the left and right edges of the active video also needs real-time RGB data. Since this task uses the blank levels and maximum values obtained in BLK1, it can only be executed on the second frame in BLK2. The task of searching high pixels should also be assigned to BLK2 since these pixels will be needed later in BLK3. When correct sampling frequency has been identified and high quality pixels become available, BLK3 can be invoked to collect data for phase calculation.

For the algorithms to function correctly, certain variables have to be passed from software to hardware, and *vice versa*. A memory unit is required for storing these variables. This memory is also responsible for storing data collected by BLK3. Therefore, two additional hardware blocks are needed: BLK4 of a memory controller and BLK5 of a memory of certain size.

Software Development Guide

In real application, the algorithms can be implemented as an "autosync" function in digital display devices. When the user pushes the autosync button an interrupt request is presented to the microcontroller. If granted, the interrupt handler dedicated for this autosync function will be invoked. The action sequence that should be coded in this function is shown later:

(1) Set initial sampling frequency to the value defined in the VESA specification.
(2) Find blank levels and maximum RGB values, and store them in memory (hardware BLK1).
(3) Calculate threshold; find active video edges; find high-quality pixels, and store the result in memory (hardware BLK2).
(4) Compute HADRM (software in microcontroller).
(5) Is HADRM=HADR true? If not, calculate DR(new) and reset frequency register; go to step 2. If yes, then the correct sampling frequency is found; go to the next step (software in microcontroller).
(6) Set phase register, collect data for this phase. Repeat this step for all available phases (hardware BLK3).
(7) Do calculation to find the desired phase (software in microcontroller).

Estimation of Execution Time

The execution times for BLK1 and BLK2 are both one frame of time. BLK3 requires 32 frames if there are 32 phases. The time required for software activity is depended upon the MIPS of the selected microcontroller and the parameter chosen for measuring image quality.

CONCLUSION

A method of automatically searching the sampling frequency and the sampling phase for graphic digitizer is presented in this article. Both the frequency searching and phase searching algorithms have been intensively tested and the test result has been presented. The frequency searching algorithm can efficiently adjust the PLLs divider ratio and correctly recover the encoded image. For the search of the desired sampling phase, several parameters have been introduced to measure the quality of the recovered image. To quantify the quality of these images, our conclusion is that any one of the three parameters (*first derivative*, *second derivative*, and *distance*) can be used. Compared to *first derivative*, *second derivative* can calculate the flatness to the second order. But it is also the most computation-intensive parameter. In terms of memory usage and CPU computational burden, parameter *distance* is the most efficient one. The algorithms can be applied in any application that requires the effort of choosing sampling frequency and sampling phase for ADC converter. Especially, in application of digital display device (a display of fixed pixel structure such as LCD, PDP, FED, EL, DMD, and etc.) driven by analog video/graphic source, these algorithms can be implemented as an autosync function.

ACKNOWLEDGMENTS

The authors thank Ellen Chen, Undrea Fields, Michael Railey, and Candel Ramirez for their help in this project. The authors give special thanks to Mike Fleischer for providing the mathematic model and programming support.

REFERENCES

[1] THS8083 Data Manual, Texas Instruments Inc., June 2000.
[2] AD9884A Data Sheet, Rev. B, Analog Devices, Inc., 2000.
[3] VESA and Industry Standards and Guidelines for Computer Display Monitor Timing, Version 1.0, Reversion 0.8, VESA, Sep. 17, 1998.
[4] VESA Generalized Timing Formula Standard, Version 1.1, VESA, Sep. 2, 1999.
[5] H. Mair and L. Xiu, "An architecture of high performance frequency and phase synthesis", IEEE J. Solid-State Circuits **35**, 835 (2000).
[6] A. J. Eglit, "Method and apparatus implemented in a computer system for determining the frequency used by a graphics source for generating an analog display signal", US Patent No. 5,847,701 (1998).
[7] T. Nakano, "Automatic image quality adjustment device adjusting phase of sampling clock for analog video signal to digital video signal conversion", US Patent No. 6,097,444 (2000).
[8] M. Shinagawa, "Jitter analysis of high-speed sampling systems", IEEE J. Solid-State Circuits **25**, 220 (1990).
[9] M. G. Makhija and V. P. Telang, "Simulating clock jitter in digital communication systems", *IEEE International Conference on Communications, ICC 96*, Vol. **2**, (IEEE, 1996) pp. 716–720.
[10] S. Omori and K. Ueda, "High-resolution image using several sampling-phase shifted images-density doubling of single chip color CCD with pixel-shift technique", *International Conference on Consumer Electronics, ICCE 2000 Dig. Tech. Papers*, (ICCE, Los Angeles, CA, 2000) pp. 178–179.
[11] A. J. Eglit, "Method and apparatus implemented in an automatic sampling phase control system for digital monitors", US Patent No. 6,268,848 (2001).
[12] A. J. Eglit, "Digital display unit which adjusts the sampling phase dynamically for accurate recovery of pixel data encoded in an analog display signal", US Patent No. 6,483,447 (2002).
[13] User Manual, TLA700 Series Logical Analyzer, Version 3.0, 071-0265-00, (Tektronix, Beaverton, OR).
[14] User Manual, Tektronix Logic Analyzer Family, Version 3.2 Software, 071-0729-00, (Tektronix, Beaverton, OR).