

6555x

HiQVideo™ Series
Programming Examples

Application Note
Revision 1.2

May 1997

P R E L I M I N A R Y

CHIPS®

Copyright Notice

Copyright© 1997 Chips and Technologies, Inc. ALL RIGHTS RESERVED.

This manual is copyrighted by Chips and Technologies, Inc. You may not reproduce, transmit, transcribe, store in a retrieval system, or translate into any language or computer language, in any form or by any means - electronic, mechanical, magnetic, optical, chemical, manual, or otherwise - any part of this publication without the express written permission of Chips and Technologies, Inc.

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 252.277-7013.

Trademark Acknowledgment

CHIPS Logo is a registered trademark of Chips and Technologies, Inc.

HiQVideo, HiQV32, trademarks of Chips and Technologies, Inc.

VESA is a registered trademark of Video Electronics Standards Association.

All other trademarks are the property of their respective holders.

Disclaimer

This document provides general information for the customer. Chips and Technologies, Inc., reserves the right to modify the information contained herein as necessary and the customer should ensure that it has the most recent revision of the document. CHIPS makes no warranty for the use of its products and bears no responsibility for any errors which may appear in this document. The customer should be on notice that many different parties hold patents on products, components, and processes within the personal computer industry. Customers should ensure that their use of the products does not infringe upon any patents. CHIPS respects the patent rights of third parties and shall not participate in direct or indirect patent infringement.

Revision History

<u>Revision</u>	<u>Date</u>	<u>By</u>	<u>Comment</u>
1.0	8/96	RJ/lc	Official Release
1.1	11/96	CT/lc	Added "Confidential" Markings and updated the disclaimer page.
1.2	5/13/97	LC	Removed NDA Requirements

Table of Contents

1.0 Introduction	1
2.0 Video Playback through PCI/VL Bus.....	1
3.0 Video Capture and Playback Through Video Port	2
3.1 VideoRect	2
3.2 CropRect.....	3
3.3 ZoomUp.....	3
3.4 Panning	4
3.5 Scaling	4
4.0 Sample Programs	5
4.1 Video Playback Using the Host CPU Bus	6
4.1.1 How to enable video playback module (Init)	6
4.1.2 How to disable video playback module (Exit)	6
4.1.3 How to compute alignment factor for video playback window (Init).....	7
4.1.4 How to enable/disable video playback (XY Keying)	9
4.1.5 How to program video playback buffer pitch (stride)	9
4.1.6 How to set video playback color format.....	10
4.1.7 How to program video playback window and buffer address	11
4.1.8 How to program color key registers (ColorDepth dependent)	14
4.1.9 How to enable/disable color key	15
4.2 Video Capture Using the Video Port	16
4.2.1 How to enable video capture and playback module (Init)	17
4.2.2 How to disable video playback and capture module (Exit).....	18
4.2.3 How to start video capture	18
4.2.4 How to stop video capture.....	19
4.2.5 How to set input video color format.....	20
4.2.6 How to set interlaced or non-interlaced video input.....	20
4.2.7 How to enable/disable double buffer.....	21
4.2.8 How to scale input video (before acquiring into frame buffer)	22
4.2.9 How to crop input video (programming of acquisition rectangle).....	23

HiQVideo™ Series Programming Examples

1.0 Introduction

This application note shows how the CHIPS HiQVideo™ Series controllers can be used for video capture and playback. This document includes a description of the hardware configuration, a discussion of the functions, and actual programming examples.

2.0 Video Playback through PCI/VL Bus

The new generation of Chips and Technologies, Inc. Multimedia Accelerators (6555x) supports Color Space Conversion and Stretching (Zooming) in the back end with the chroma color key. The color space conversion functionality of the 6555x can be made available to video codecs by implementing the off-screen surface support in the DCI Provider (Windows 3.1 drivers). Only the playback feature of 6555x multimedia module is used to implement extended DCI functionality. This means the video input to the 6555x is kept in the frozen state (not grabbing) when DCI is running. For video playback, the CPU can write YUV, RGB15, and RGB16 data into the off-screen memory and fill the destination rectangle (where video need to be displayed on the visible screen) with the color key. Video can be zoomed up if the destination rectangle is bigger than the source rectangle in the off-screen buffer.

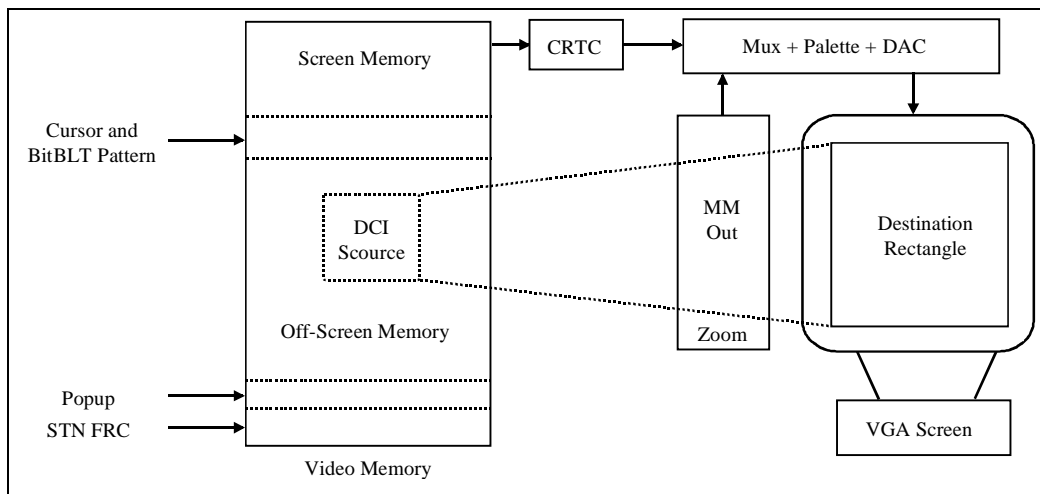


Figure 1 Video Playback with zoom for DCI Off-Screen Implementation

CHIPS Multimedia Accelerators currently supports the following surface formats:

FourCC Code	Interpretation	Byte Order (0-3)
YUY2	default YUV 4:2:2	Y0 U0 Y1 V0
YVYU	Swap U&V in YUV 4:2:2	Y0 V0 Y1 U0
RGB15	RGB 5-5-5	
RGB16	RGB 5-6-5	

3.0 Video Capture and Playback Through Video Port

The new generation of CHIPS Multimedia Accelerators (6555x) can also capture live video from the video port into the off-screen memory and play it back with color space conversion onto a color keyed destination rectangle on the visible screen. Playback video can be zoomed up to fill the bigger destination rectangle while incoming video can be scaled down to fit into a smaller off-screen memory buffer or smaller destination rectangle. Zoomed video can be smoothed out with horizontal and vertical interpolation. Scaled down video can also be filtered out at input before capturing into the frame buffer. Input video can be cropped for the extra data which is usually associated with the NTSC or PAL video. The 6555x hardware can accommodate fast or slow capture applications through the CPU Bus by capturing the video frames in one of three methods: continuously, one frame at a time, or one every nth ($n = 1-15$) frame. Following diagram demonstrates the video capture.

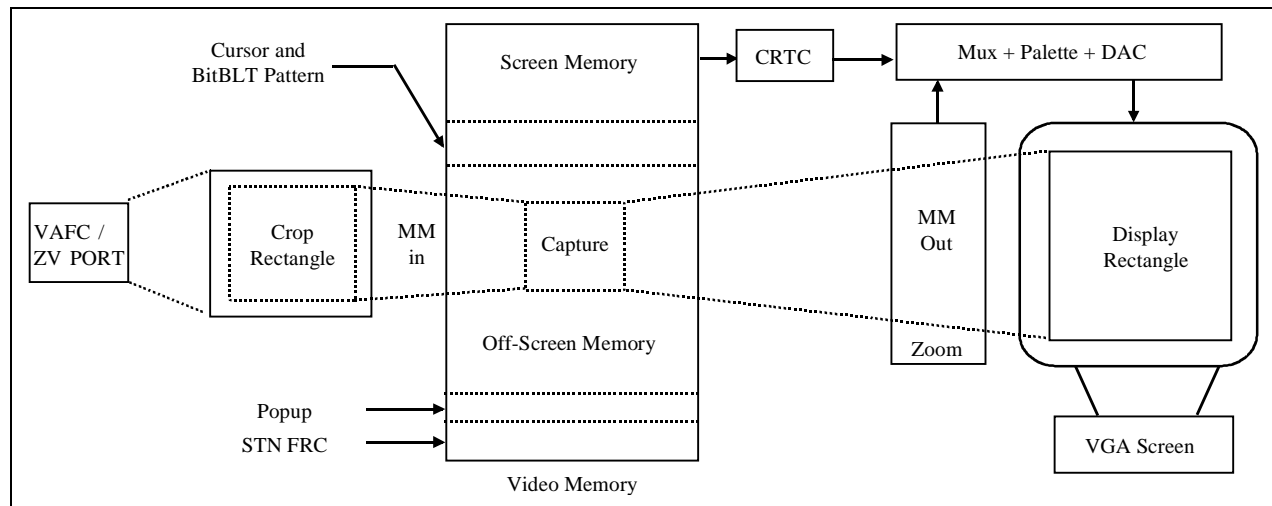


Figure 2: Zoom Input Video and Playback

Input video is scaled down and the playback is zoomed up to fit into display window

3.1 VideoRect

VideoRect comes from the input video stream fed through the Video Port (VAFC / ZV Port) and comprises of one of the following sizes based on the input source.

- NSTC: 640x480 60 fields / second (interlaced), Square Aspect Ratio (4:3).
720x486 60 fields / second (interlaced), Non-square Aspect Ratio.
- PAL: 768x576 50 fields / second (interlaced), Square Aspect Ratio (4:3).
720x576 50 fields / second (interlaced), Non-square Aspect Ratio.
- MPEG1: 320x240 30 frames / sec (non-interlaced), Square Aspect Ratio (4:3).
352x288 30 frames / sec (non-interlaced), Non-square Aspect Ratio.

Top-left of VideoRect is always at (0,0).

3.2 CropRect

CropRect is defined relative to the **VideoRect**. **CropRect** is used to crop off some pixels from the top, left, right, or bottom to fit the image into a square pixel ratio or to drop some unwanted pixels. **CropRect** is programmed using the acquisition window registers.

After cropping, the video is scaled down to fit into a smaller memory buffer or in a smaller display window. The scaled video is captured into off-screen video memory buffer or buffers (as in double buffer mode). There is a horizontal filter to reduce the sampling artifacts caused by input video scaling. Video in the capture buffer is displayed on top of the pixels which matches the color key and/or with a specified rectangular window.

3.3 ZoomUp

If client area of a window (**DispRect**) is larger than the capture buffer rectangle (**CaptRect**), the video can be zoomed up to fit into the DispRect.

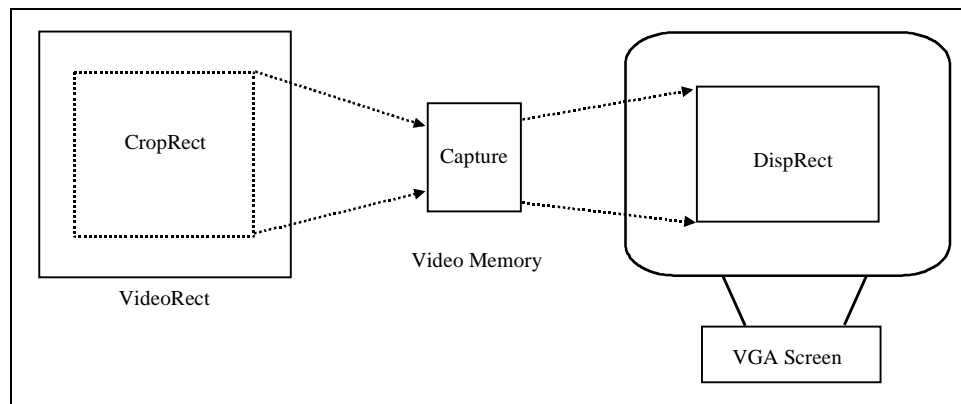


Figure 3: Video is scaled to fit into Input Display Window (No Zoom)

3.4 Panning

If client area of the window is smaller than the capture buffer rectangle, the display window can be panned over the bigger capture buffer. It is also possible to zoom-up the video and then crop the display window size and pan it over the bigger zoomed video window.

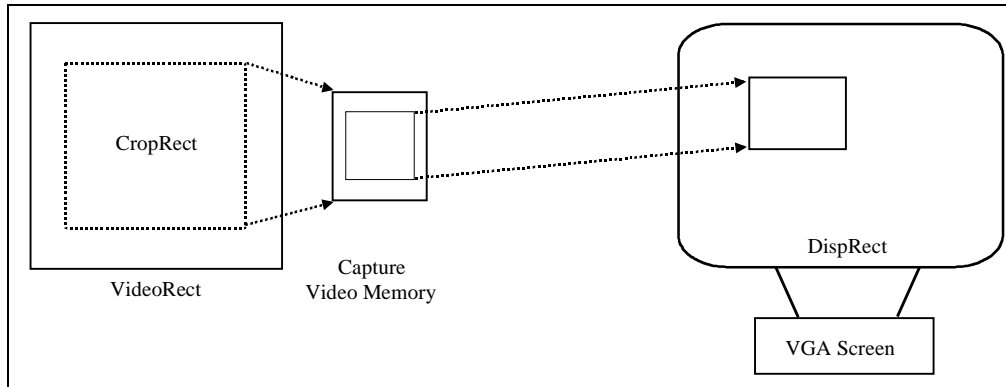


Figure 4: Small DispRect is panned over larger CaptureRect (No Zoom)

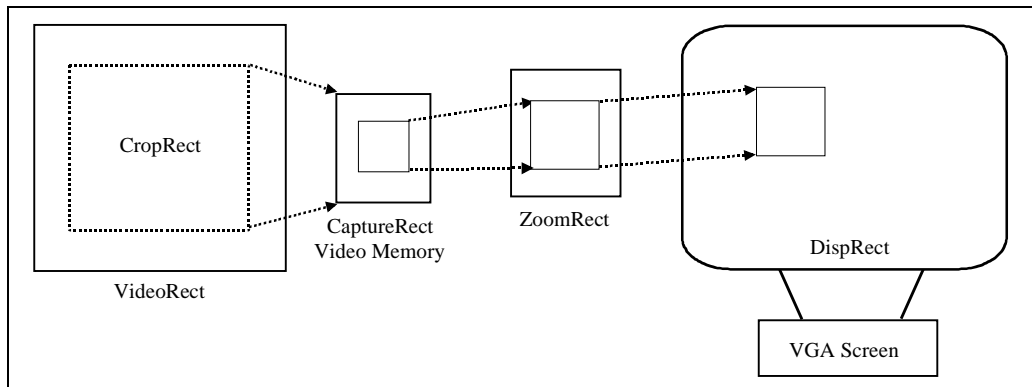


Figure: 5 Small DispRect is panned over larger zoomed CaptureRect

3.5 Scaling

Input Video Scaling can also be used to fit a bigger input VideoRect into a smaller displayed window (DispRect).

As shown in the previous diagrams, video playback through the PCI bus or video capture through the video port (Zoom Video Port) is done in the off-screen memory. Availability of the off-screen memory should be handled through the display (GUI) driver.

4.0 Sample Programs

The following sections (4.1 and 4.2) show sample code for programming the video playback using host CPU bus and video capture using the video port.

We assume the following variables in the programming examples.

```
XR?? = Extended Registers at 3d6/3d7.
MR?? = Multimedia Registers at 3d2/3d3.
SAR?? = Sequencer Arbitration Registers.

ulXrAddr = 0x3d6;           // 55x Extended Register Addr for as VGA Io
          = MmioBase + 2*0x3d6; // Register offset using MMIO
ulMrAddr = 0x3d2;           // 55x Multimedia register addr as VGA Io
          = MmioBase + 2*0x3d2; // Register offset using MMIO
ulCrAddr = 0x3d4;           // 55x CRT Registers Addr as VGA Io
          = MmioBase + 2*0x3d4; // Register offset using MMIO
ulFrAddr = ulIoBase + 0x3d0; // 55x Flat Panel Register Addr as VGA Io
          = MmioBase + 2*0x3d0; // Register offset using MMIO

osbAddress = Address of the OffScreen Buffer
osbWidthBytes = WidthBytes (pitch) of the OffScreen Buffer
VideoDisplaySkew_X = x-alignment to be added to screen x coordinate
VideoDisplaySkew_Y = y-alignment to be added to screen y coordinate
DisplayPanningStart_X = Display Panning Left (0 if no panning)
DisplayPanningStart_Y = Display Panning Top (0 if no panning)
ScreenWidth = Width of Screen in Pixels.
ScreenHeight = Height of the screen in scanlines.
ScreenBitsPixel = Colordepth for screen
ScreenFlags = contains 5-5-5/5-6-5 information for 16BPP modes
```

4.1 Video Playback Using the Host CPU Bus

Video Playback Using the Host CPU Bus

- How to enable video playback module (Init)
- How to disable video playback module (Exit)
- How to compute alignment factor for video playback window (Init)
- How to enable/disable video playback (XY Keying)
- How to program video playback buffer pitch (stride)
- How to set video playback color format
- How to program video playback window and buffer address
- How to program color key registers (ColorDepth dependent)
- How to enable/disable color key

4.1.1 How to enable video playback module (Init)

1. Save and Set XRD0[4] = 1
2. Save and Set SAR04 = 0x2A; // To get wider (> 352) playback buffer width

```
static USHORT      XRD0,SAR04;
Init_Playback()
{
    UCHAR          XR_Index;
    XR_Index = ReadPortUshort(ulXrAddr);          // Save XR Index
    WritePortUchar(ulXrAddr,0xd0);                // Read XRD0
    xrD0 = ReadPortUshort(ulXrAddr);              // Save XRD0
    WritePortUshort(ulXrAddr,xrD0 | 0x1000);      // Enable video playback module

    WritePortUshort(ulXrAddr,0x044e);             // Read SAR04
    WritePortUchar(ulXrAddr,0x4f);
    SAR04 = ReadPortUshort(ulXrAddr);             // Save SAR04
    WritePortUshort(ulXrAddr,(SAR04 & 0x00ff) | 0x2a00); // SAR04=2a

    WritePortUchar(ulXrAddr,XR_Index);           // Restore XR_Index
}

```

4.1.2 How to disable video playback module (Exit)

1. Restore XRD0.
2. Restore SAR04.

```
Playback_Exit()
{
    UCHAR          XR_Index;

    XR_Index = ReadPortUshort(ulXrAddr);          // Save XR Index
    WritePortUshort(ulXrAddr,XRD0);              // Restore XRD0
    WritePortUshort(ulXrAddr,0x044e);            // Read SAR04
    WritePortUshortAddr,SAR04);                  // Restore SAR04
    WritePortUchar(ulXrAddr,XR_Index);           // Restore XR_Index
}

```

4.1.3 How to compute alignment factor for video playback window (Init)

In the 6555x, the zero position of video playback rectangle does not align with the zero of the display controller. There is an offset in both the X and Y directions. This alignment factor is dependent on the resolution, color depth, display type, and refresh rate. The alignment factor can be computed from the CRT and panel timing registers with some exception in interlaced modes. The following code shows the alignment computation.

```
//-----
// CalculateDisplaySkew() - Computes x-y display skew. Must be called after
// mode set, display switch or refresh rate change. It also takes care of CRT
// or simultaneous mode. It programs MR1E to interlaced or non-interlaced
// mode.
// Updates VideoDisplaySkew_X, VideoDisplaySkew_Y variables with new skew.
//-----
void CalculateDisplaySkew()
{
int    h_total,h_blank_start;
int    v_total,v_sync_end,cr07;
int    cr_index,fr_index;
int    PanelSizeX,PanelSizeY;
int    RefreshRate,mr1e;

    RefreshRateBit = 0; // assume non-interlaced mode or panel
    fr_index = ReadPortUchar(ulFrAddr); // save FR index
    WritePortUchar(ulFrAddr,1); // Read FR01
    if((ReadPortUshort(ulFrAddr) & 0x0300) != 0x0100)
    { // Panel or Simultaneous mode (Not CRT Mode) (FR01[1-0] == 0x01)
//
// VideoDisplaySkew_X = (fh_total + 3 - fh_display_enable_end)*8 -1;
// VideoDisplaySkew_X = (FR23 + 3 - FR20)*8 -1;
//
        WritePortUchar(ulFrAddr,0x23); // read FR23 (h_total)
        h_total = (ReadPortUshort(ulFrAddr) >> 8) & 0x00ff; // ignoring overflow
        WritePortUchar(ulFrAddr,0x20); // read FR20 (h_display_enable_end)
        h_blank_start = (ReadPortUshort(ulFrAddr) >> 8) & 0x00ff;
        VideoDisplaySkew_X = (h_total - h_blank_start +3)*8 -1;
//
// VideoDisplaySkew_Y = (fv_total - fv_sync_end);
// VideoDisplaySkew_Y = FR36[3-0]:FR33 -((FR35[7-4]:FR31 & 0xffff0) | FR32 & 0x0f);
//
        WritePortUchar(ulFrAddr,0x33); // read FR33 (v total)
        v_total = (ReadPortUshort(ulFrAddr) >> 8) & 0x00ff;
        WritePortUchar(ulFrAddr,0x31); // read FR31 (v_sync_start)
        v_sync_end = (ReadPortUshort(ulFrAddr) >> 8) & 0xf0;
        // lower 4 bits from sync_end reg
        WritePortUchar(ulFrAddr,0x32); // read FR32[3-0] (v_sync_end)
        v_sync_end |= ((ReadPortUshort(ulFrAddr) >> 8) & 0x0f);
//
// Collect overflow bits of vertical registers.
//
        WritePortUchar(ulFrAddr,0x36); // read FR36[3-0] (overflow v_total bits)
        v_total |= (ReadPortUshort(ulFrAddr) & 0x0f00);
        WritePortUchar(ulFrAddr,0x35); // read FR35[7-4] (overflow v_sync_end bits)
        v_sync_end |= (ReadPortUshort(ulFrAddr) >> 4) & 0x0f00;

        VideoDisplaySkew_Y = (v_total - v_sync_end);

        PanelSizeX = 8*(h_blank_start+1);
        if(PanelSizeX > ScreenWidth)
        { // mode size is less than panel size, we assume cenetring enabled
            // We do not supoport stretching at this time.
            VideoDisplaySkew_X += (PanelSizeX - ScreenWidth)/2;
            WritePortUchar(ulFrAddr,0x30); // read FR30 (v_dee, panelSize)
        }
    }
}
```

```

        PanelSizeY = (ReadPortUshort(ulFrAddr) >> 8) & 0x00ff;
        WritePortUchar(ulFrAddr, 0x35);
        // read FR35[3-0] (v_dee overflow bits, panelSize)
        PanelSizeY |= (ReadPortUshort(ulFrAddr) & 0x0f00);
        PanelSizeY++; // PanelSizeY = FR35[3-0]:FR30 + 1
        VideoDisplaySkew_Y += (PanelSizeY - ScreenHeight)/2;
    }
}
else
{ // CRT Mode (not Panel)
//
// VideoDisplaySkew_X = (h_total + 3 - h_blank_start)*8 -1;
//
    cr_index = ReadPortUchar(ulCrAddr); // Save CR Index
    WritePortUchar(ulCrAddr, 0x00); // read CR00 (h total)
    h_total = (ReadPortUshort(ulCrAddr) >> 8) & 0x00ff;
    WritePortUchar(ulCrAddr, 0x02); // read CR02 (h blank start)
    h_blank_start = (ReadPortUshort(ulCrAddr) >> 8) & 0x00ff;
    VideoDisplaySkew_X = (h_total - h_blank_start + 3)*8 -1;
//
// VideoDisplaySkew_Y = (v_total - v_sync_end);
//
    WritePortUchar(ulCrAddr, 0x06); // read CR06 (v total)
    v_total = (ReadPortUshort(ulCrAddr) >> 8) & 0x00ff;
    WritePortUchar(ulCrAddr, 0x10); // read CR10 (v sync start)
    v_sync_end = (ReadPortUshort(ulCrAddr) >> 8) & 0xf0;
    // lower 4 bits from sync_end reg
    WritePortUchar(ulCrAddr, 0x11); // read CR11[3-0] (v sync end)
    v_sync_end |= ((ReadPortUshort(ulCrAddr) >> 8) & 0x0f);
//
// Collect overflow bits of vertical registers.
//
    WritePortUchar(ulCrAddr, 0x07); // read CR07 (overflow bits)
    cr07 = (ReadPortUshort(ulCrAddr) >> 8) & 0x00ff;
    if(cr07 & 0x01) v_total |= 0x100; // cr07[0] = 8th bit of v_total
    if(cr07 & 0x20) v_total |= 0x200; // cr07[5] = 9th bit of v_total
    if(cr07 & 0x04) v_sync_end |= 0x100; // cr07[2] = 8th bit of v_sync_start
    if(cr07 & 0x80) v_sync_end |= 0x200; // cr07[7] = 9th bit of v_sync_start
    WritePortUchar(ulCrAddr, 0x30); // read CR30 (extended v_total bits)
    v_total |= (ReadPortUshort(ulCrAddr) & 0x0700);
    WritePortUchar(ulCrAddr, 0x32); // read CR32 (extended v_sync_start bits)
    v_sync_end |= (ReadPortUshort(ulCrAddr) & 0x0700);

    VideoDisplaySkew_Y = (v_total - v_sync_end);
//
// Read interlaced bit from CR70[7]
//
    WritePortUchar(ulCrAddr, 0x70); // read CR70[7]
    if(ReadPortUshort(ulCrAddr) & 0x8000)
    { // interlaced CRT mode (1024x768)
        RefreshRateBit |= 1;
        if(ScreenWidth == 1024)
        { // 1024x768x43Hz (interlaced)
            VideoDisplaySkew_Y += 5; // why +5? (don't know)
        }
        else
        { // 1280x1024x43Hz
            VideoDisplaySkew_Y = 2*(VideoDisplaySkew_Y);
        }
    }
    WritePortUchar(ulCrAddr, cr_index); // Restore CR Index
}
WritePortUchar(ulFrAddr, fr_index); // Restore FR index
//
// Set Video Playback to Interlaced Mode

```

```
//
WritePortUchar(ulMrAddr, MR_VDP_CTRL_1);    // read MR1E
mr1e = ReadPortUshort(ulMrAddr);           // Read current value
mr1e &= ~(VDC1_INTERLACE << 8);            // assume non-interlaced graphics mode
if(value == TRUE) mr |= (VDC1_INTERLACE << 8);    // interlace graphics mode
WritePortUshort(ulMrAddr, mr1e);           // write new value
}
```

4.1.4 How to enable/disable video playback (XY Keying)

```
//-----
// EnableVideo() : Turns on/off the display of video overlay.
//
// Enter:
//     BOOL  bEnable = FALSE : Turns off the video overlay.
//           bEnable = TRUE  : Turns on the video overlay.
// Exit :
//     Nothing
//-----
void EnableVideo(BOOL bEnable)
{
int    mr3c;

WritePortUchar(ulMrAddr, MR_VDP_CKEY_CTRL);
mr3c = (int)ReadPortUshort(ulMrAddr);    // Read current value
mr3c &= ~(VDC_EV_OVERLAY | VDC_EV_XY_RECT) << 8);
// assume video overlay off
if(bEnable != FALSE) mr3c |= ((VDC_EV_OVERLAY | VDC_EV_XY_RECT) << 8);
// turn on overlay
WritePortUshort(ulMrAddr, mr3c);         // write new value
}
```

4.1.5 How to program video playback buffer pitch (stride)

```
//
// Set pitch of the frame buffer rect for VideoInput and VideoDisplay
//
pitch_qd = (((osbWidthBytes >> 3) - 1) << 8);
WritePortUshort(ulMrAddr, pitch_qd | MR_VIN_PITCH_QD);    // for Capture
WritePortUshort(ulMrAddr, pitch_qd | MR_VDP_PITCH_QD);    // for Playback
```

4.1.6 How to set video playback color format

CHIPS 6555x products support three basic color formats for video playback: YUV4:2:2, RG555, and RGB565. Each format is 16 bit per pixel. The 6555x also allows swapping the UV positions within a 32 bit dword. The default sequence for YUV4:2:2 is Byte0=Y0, Byte1=U, Byte2=Y1, Byte3=V. Luma (Y) is unsigned (0-255) and U,V can be unsigned or signed (2's complement). This way the 6555x can support two RGB16 formats and four YUV4:2:2 formats (YUYV, YVYU: swapped, YUYVs: signed, YVYUs: signed, swapped). The following code shows the video playback format selection.

```
//-----
// SetVideoDisplayFormat() - sets Video Storage Format bits
//-----
void SetVideoDisplayFormat(int iVideoFormat)
{
    UINT i,cf;

    WritePortUchar(ulMrAddr,MR_VDP_CTRL_2); // read video display control reg 2
    i = ReadPortUshort(ulMrAddr);
    i &= ~(VDC2_FORMAT << 8); // clear format bits (0 is YUV normal)
    switch(iVideoFormat)
    {
        case CMM_FMT_YUV_422: // YUYV (YUV4:2:2 normal)
            i |= (VDC2_YUV422 << 8);
            break;
        case CMM_FMT_UV_SWAP: // YVYU (YUV4:2:2 uv swap)
            i |= (VDC2_YUV422_UVS << 8);
            break;
        case CMM_FMT_SIGNED_UV: // YUYV (YUV4:2:2 signed UV)
            i |= (VDC2_YUV422_SUV << 8);
            break;
        case CMM_FMT_SIGNED_UV_SWAP: // YUYV (YUV4:2:2 signed UV swapped)
            i |= (VDC2_YUV422_UV << 8);
            break;
        case CMM_FMT_RGB_555: // RGB16 5-5-5
            i |= (VDC2_RGB555 << 8);
            break;
        case CMM_FMT_RGB_565: // RGB16 5-6-5
            i |= (VDC2_RGB565 << 8);
            break;
        default:
            return ;
    }
    WritePortUshort(ulMrAddr,i); // write new format
}
```

4.1.7 How to program video playback window and buffer address

```
//-----
// SetDisplayRect() : Sets display window Position and Size. Rectangle may
//                    span beyond screen. Needs to be clipped.
//
//
//      +-----+
//      | Video   |
//      | Clip   | +-----+
//      |  | Vis | |
//      +-----+
//
//      |
//      | Screen
//      |
//      +-----+
//
// Enter:
//      lpRect      Display Rectangle (client rectangle)
//      DisplayPanningStart_X, DisplayPanningStart_Y : Panning Rect Start (0:NoPan)
//      VideoDisplaySkew_X, VideoDisplaySkew_Y contains skew factors.
//      ScreenWidth and ScreenHeight are screen resolution.
//      osbMemAddress is ofscreen buffer address (relative to frame buffer)
//      osbBytesPerPixel is bytes per pixel for video data (2)
//      osbWidthBytes is the video buffer width in bytes (pitch)
//      InterpolX is the X-Position below which interpolation would not be enabled
//      rSrcRect is a source rectangle (left,top,right,bottom) relative to the
//      video buffer.
// Exit :
//      Nothing
//-----
void SetDisplayRect(LPRECTL lpRect)
{
    int    xLeft,yTop,xRight,yBottom;
    int    wSrc,wDst,hSrc,hDst;
    int    xClip = 0, yClip = 0;
    int    xSrc,ySrc;
    UINT    mr2a,mr2b,mr2c,mr2d,mr2e,mr2f,mr30,mr31;
    union WORD16      u;
    union WORD32      ud;
    UINT    mr1e,mr1f;
    UINT    zoom_max = VDP_ZOOM_X_MAX;           // change it for older revs of 550
    UINT    zoom_x,zoom_y;
    ULONG    SrcOffset;

    ud.d = osbMemAddress;                        // memory address to be programmed
    wSrc = (int)rSrc.right - (int)rSrc.left;
    wDst = (int)rDisp.right - (int)rDisp.left;
    hSrc = (int)rSrc.bottom - (int)rSrc.top;
    hDst = (int)rDisp.bottom - (int)rDisp.top;
    xSrc = (int)rSrc.left;
    ySrc = (int)rSrc.top;
    if((xLeft = (int)lpRect->left - DisplayPanningStart_X) < 0)
    { // need left clipping
        xClip = -xLeft;           // apply clipping
        xLeft = 0;
        if(wSrc < wDst)
        { // zoom in x-direction is in place. calculate xClipSrc from dest
            xClip = (int)(((ULONG)xClip*(ULONG)wSrc)/(ULONG)wDst);
        }
        xSrc += xClip;
    }
    if((yTop = (int)lpRect->top - DisplayPanningStart_Y) < 0)
    { // need top clipping
```

```

    yClip = -yTop;
    yTop = 0;
    if(hSrc < hDst)
    { // zoom in x-direction is in place. calculate xClipSrc from dest
        yClip = (int)(((ULONG)yClip*(ULONG)hSrc)/(ULONG)hDst);
    }
    ySrc += yClip;
}
SrcOffset = (xSrc*osbBytesPerPixel + (ULONG)ySrc*osbWidthBytes);
ud.d += SrcOffset;

xRight = (int)lpRect->right - DisplayPanningStart_X;
if(xRight >= (int)ScreenWidth)
    xRight = (int)(ScreenWidth); // clip right
yBottom = (int)lpRect->bottom - DisplayPanningStart_Y;
if(yBottom > (int)ScreenHeight)
    yBottom = (int)ScreenHeight;

u.w = (USHORT)(((int)VideoDisplaySkew_X + xLeft);
mr2a = ((UINT)u.b[0] << 8) + MR_VDP_WIN_XL_L;
mr2b = ((UINT)u.b[1] << 8) + MR_VDP_WIN_XL_H;
u.w = (USHORT)(((int)VideoDisplaySkew_X + xRight -1);
mr2c = ((UINT)u.b[0] << 8) + MR_VDP_WIN_XR_L;
mr2d = ((UINT)u.b[1] << 8) + MR_VDP_WIN_XR_H;
u.w = (USHORT)(((int)VideoDisplaySkew_Y + yTop);
mr2e = ((UINT)u.b[0] << 8) + MR_VDP_WIN_YT_L;
mr2f = ((UINT)u.b[1] << 8) + MR_VDP_WIN_YT_H;
u.w = (USHORT)(((int)VideoDisplaySkew_Y + yBottom -1);
if(ScreenWidth == 1280) u.w += 2;
// interlaced modes (don't have time to figure out)
mr30 = ((UINT)u.b[0] << 8) + MR_VDP_WIN_YB_L;
mr31 = ((UINT)u.b[1] << 8) + MR_VDP_WIN_YB_H;

WritePortUshort(ulMrAddr, mr2a); // program Left
WritePortUshort(ulMrAddr, mr2b);
WritePortUshort(ulMrAddr, mr2c); // program right
WritePortUshort(ulMrAddr, mr2d);
WritePortUshort(ulMrAddr, mr2e); // program top
WritePortUshort(ulMrAddr, mr2f);
WritePortUshort(ulMrAddr, mr30); // program bottom
WritePortUshort(ulMrAddr, mr31);

//
// Program memory address registers for buffer1
//
WritePortUshort(ulMrAddr, ((UINT)ud.b[0] << 8) | MR_VDP_ADDR_1_L); // MR22
WritePortUshort(ulMrAddr, ((UINT)ud.b[1] << 8) | MR_VDP_ADDR_1_M); // MR23
WritePortUshort(ulMrAddr, ((UINT)ud.b[2] << 8) | MR_VDP_ADDR_1_H); // MR24
ud.d = osbMemAddress2 + SrcOffset; // memory address to be programmed
WritePortUshort(ulMrAddr, ((UINT)ud.b[0] << 8) | MR_VDP_ADDR_2_L); // MR25
WritePortUshort(ulMrAddr, ((UINT)ud.b[1] << 8) | MR_VDP_ADDR_2_M); // MR26
WritePortUshort(ulMrAddr, ((UINT)ud.b[2] << 8) | MR_VDP_ADDR_2_H); // MR27

//
// Program Zoom and Interpolation bits. This function used be a separate
// routine but because of screwed-up interpolation (position dependent
// for 32 bit 65550) we have to bring this here.
//
WritePortUchar(ulMrAddr, MR_VDP_CTRL_2);
mr1f = ReadPortUshort(ulMrAddr);
mr1f &= ~(VDC2_H_INTERPOL | VDC2_V_INTERPOL | VDC2_VI_RUNAVRG) << 8);
// assume no interpolation needed

WritePortUchar(ulMrAddr, MR_VDP_CTRL_1);
mr1e = ReadPortUshort(ulMrAddr);
mr1e &= ~(VDC1_ZOOM_X | VDC1_ZOOM_Y) << 8); // assume no zoom

```



```

if(wDst > wSrc)
{ // horizontal zoom needed
    mrlf |= (VDC2_H_INTERPOL << 8);
    // enable horizontal interpolation (doesn't cost)
    zoom_x = (int)((((DWORD)wSrc*zoom_max) / (DWORD)wDst);
    WritePortUshort(ulMrAddr, (zoom_x << 8) | MR_VDP_ZOOM_X);
    mrle |= (VDC1_ZOOM_X << 8); // enable video Dstlay zoom
}
if(hDst > hSrc)
{ // vertical zoom needed
    if(!InterpolX || (osbWidthBytes <= 352*2 && xLeft >= InterpolX))
    { // we can enable vertical interpolation if there is no position
        // constraint or SrcPitch is within 352 pixels and x is more than
        // interpolation threshold.
        mrlf |= InterpolBits; // enable vertical interpolation only if meets
        // our criteria (x position)
    }
    zoom_y = (int)((((DWORD)hSrc*zoom_max) / (DWORD)hDst);
    WritePortUshort(ulMrAddr, (zoom_y << 8) | MR_VDP_ZOOM_Y);
    mrle |= (VDC1_ZOOM_Y << 8); // enable zoom in y direction
}
WritePortUshort(ulMrAddr, mrle); // set zoom factors
WritePortUshort(ulMrAddr, mrlf); // set interpolation bits
}

```

4.1.8 How to program color key registers (ColorDepth dependent)

```
//-----
// SetColorKey() : Programs key color and mask into the hardware for video
//                  overlay.
//
// Enter:
//     DWORD   KeyColor
//             = for 4BPP indexed color (4 bits)
//             = for 8BPP indexed color (8 bits)
//             = for 16BPP 3 bytes of color B0:red,B1:Green,B2:blue
//             = for 24BPP 3 bytes of color B0:red,B1:Green,B2:blue
// Exit :
//     Nothing
//-----
void ChipsVideoOverlay::SetColorKey(DWORD KeyColor)
{
    int    ckey_0,ckey_1,ckey_2,mask_0,mask_1,mask_2;
    union  WORD32 u;

    dwColorKey = u.d = KeyColor; // easy to access to bytes

    switch(ScreenBitsPixel)
    {
    case 4: // we have color key color as index for 4BPP
    case 8:
        mask_0 = MR_VDP_CKEY_M0; // compare lower 8 bits
        mask_1 = (0xff << 8) | MR_VDP_CKEY_M1;
        mask_2 = (0xff << 8) | MR_VDP_CKEY_M2; // don't compare higher 16 bits
        ckey_0 = (((UINT)((BYTE)KeyColor)) << 8) | MR_VDP_CKEY_0;
        ckey_1 = MR_VDP_CKEY_1; // program 0 instead of garbage
        ckey_2 = MR_VDP_CKEY_2; // program 0 instead of garbage
        break;
    case 16: // we can program key color as 24BPP but use mask to select
        // valid bits for comparison (b=F8,g=F8/FC,r=F8).
        mask_0 = 0x0700 | MR_VDP_CKEY_M0; // assume 16BPP mode is 5-5-5
        mask_1 = 0x0700 | MR_VDP_CKEY_M1; // use upper 5 bits of each
        mask_2 = 0x0700 | MR_VDP_CKEY_M2; // color component for compare
        if(ScreenModeFlags & SMF_16BPP_565)
        { // 16BPP mode is 5-6-5
            mask_1 = 0x0300 | MR_VDP_CKEY_M1; // use upper 6 bits for compare
        }
        goto UseFull24BPPColorKey; // same as 24BPP
    case 24:
        mask_0 = MR_VDP_CKEY_M0; // compare lower 8 bits
        mask_1 = MR_VDP_CKEY_M1; // compare next 8 bits
        mask_2 = MR_VDP_CKEY_M2; // compare hi 8 bits
    UseFull24BPPColorKey:
        ckey_0 = (((UINT)u.b[2]) << 8) | MR_VDP_CKEY_0; // blue
        ckey_1 = (((UINT)u.b[1]) << 8) | MR_VDP_CKEY_1; // green
        ckey_2 = (((UINT)u.b[0]) << 8) | MR_VDP_CKEY_2; // red
        break;
    default:
        return;
    }

    WritePortUshort(uMrAddr,mask_0); // Program Registers into h/w
    WritePortUshort(uMrAddr,mask_1); // Program mask first
    WritePortUshort(uMrAddr,mask_2);
    WritePortUshort(uMrAddr,ckey_0); // Program color key
    WritePortUshort(uMrAddr,ckey_1);
    WritePortUshort(uMrAddr,ckey_2);
}
}
```

4.1.9 How to enable/disable color key

```
//-----
// EnableColorKey() : Turns on/off the color key & xr_rect video display.
//
// Enter:
//     BOOL  bEnable = FALSE : Turns off the color key.
//           bEnable = TRUE  : Turns on the color key.
// Exit :
//     Nothing
//-----
void EnableColorKey(BOOL bEnable)
{
    int    mr3c;

    WritePortUchar(ulMrAddr, MR_VDP_CKEY_CTRL);
    mr3c = ReadPortUshort(ulMrAddr);          // Read current value
    mr3c &= ~(VDC_EV_COLOR_KEY | VDC_EV_XY_RECT) << 8;
                                           // assume video color key off
    if(bEnable != FALSE) mr3c |= ((VDC_EV_COLOR_KEY | VDC_EV_XY_RECT) << 8);
                                           // turn on color key
    WritePortUshort(ulMrAddr, mr3c);          // write new value
}
```

4.2 Video Capture Using the Video Port

We need some additional functions to manage video capture through video port. Some of the initialization and exit code can be merged together with the playback code. Capture code should also include the previously described playback code.

- How to enable video capture and playback module (Init)
- How to disable video playback and capture module (Exit)
- How to start video capture
- How to stop video capture
- How to set input video color format
- How to set interlaced or non-interlaced video input
- How to enable/disable double buffer
- How to scale input video (before acquiring into frame buffer)
- How to crop input video (programming of acquisition rectangle)

4.2.1 How to enable video capture and playback module (Init)

This code should be executed before video starts flowing into the port.

```

1. Save and Set XRD0[4] = 1
2. Save and Set SAR04 = 0x2A; // To get wider (> 352) playback buffer width
Static USHORT      XR60,XRD0,SAR04;
CaptureInit()
{
    UCHAR          XR_Index;

    bVideoFlowingIn = 0;          // assume video is not flowing into the port
    XR_Index = ReadPortUshort(ulXrAddr); // Save XR Index
    WritePortUchar(ulXrAddr,0xd0); // Read XRD0
    xrD0 = ReadPortUshort(ulXrAddr); // Save XRD0
    WritePortUshort(ulXrAddr,xrD0 | 0x7000); // Enable video playback/Capture module

//
// Enable video port in 55x for ZV Port Style Video
//
    WritePortUchar(ulXrAddr,0x60); // Read XR60
    xr60 = ReadPortUshort(ulXrAddr); // Save XRD0
    WritePortUshort(ulXrAddr,xr60 | 0x0300);

//
// Program 55x for playback of wider (> 352) video buffer.
//
    WritePortUshort(ulXrAddr,0x044e); // Read SAR04
    WritePortUchar(ulXrAddr,0x4f);
    SAR04 = ReadPortUshort(ulXrAddr); // Save SAR04
    WritePortUshort(ulXrAddr,(SAR04 & 0x00ff) | 0x2a00); // SAR04=2a
    WritePortUchar(ulXrAddr,XR_Index); // Restore XR_Index

//
// Set video capture buffer address for both buffers.
//
    u.d = osbMemAddress; // assign to a DWORD union to access bytes
    WritePortUshort(ulMrAddr,((UINT)u.b[0] << 8) | MR_VIN_ADDR_1_L); //mr06
    WritePortUshort(ulMrAddr,((UINT)u.b[1] << 8) | MR_VIN_ADDR_1_M); //mr07
    WritePortUshort(ulMrAddr,((UINT)u.b[2] << 8) | MR_VIN_ADDR_1_H); //mr08
    WritePortUshort(ulMrAddr,((UINT)u.b[0] << 8) | MR_VIN_ADDR_2_L); //mr09
    WritePortUshort(ulMrAddr,((UINT)u.b[1] << 8) | MR_VIN_ADDR_2_M); //mr0a
    WritePortUshort(ulMrAddr,((UINT)u.b[2] << 8) | MR_VIN_ADDR_2_H); //mr0b
    WritePortUshort(ulMrAddr,((UINT)u.b[0] << 8) | MR_VDP_ADDR_1_L); //mr22
    WritePortUshort(ulMrAddr,((UINT)u.b[1] << 8) | MR_VDP_ADDR_1_M); //mr23
    WritePortUshort(ulMrAddr,((UINT)u.b[2] << 8) | MR_VDP_ADDR_1_H); //mr24
    WritePortUshort(ulMrAddr,((UINT)u.b[0] << 8) | MR_VDP_ADDR_2_L); //mr25
    WritePortUshort(ulMrAddr,((UINT)u.b[1] << 8) | MR_VDP_ADDR_2_M); //mr26
    WritePortUshort(ulMrAddr,((UINT)u.b[2] << 8) | MR_VDP_ADDR_2_H); //mr27

//
// Set Aquisition rectangle to NULL (Left=-1, right=0) to avoid capturing of first
// frame. This is needed to latch the capture counter with the new address.
//
    WritePortUshort(ulMrAddr,0xff0e); // program Left=-1
    WritePortUshort(ulMrAddr,0xff0f);
    WritePortUshort(ulMrAddr,0x0010); // program right=0
    WritePortUshort(ulMrAddr,0x0011);
    WritePortUshort(ulMrAddr,0x0012); // program top=0
    WritePortUshort(ulMrAddr,0x0013);
    WritePortUshort(ulMrAddr,0x0014); // program bottom=0
    WritePortUshort(ulMrAddr,0x0015);
}

```

4.2.2 How to disable video playback and capture module (Exit)

1. Restore XRD0.
2. Restore SAR04.

```

UCHAR      XR_Index;

XR_Index = ReadPortUshort(ulXrAddr);      // Save XR Index
WritePortUshort(ulXrAddr,XRD0);           // Restore XRD0
WritePortUshort(ulXrAddr,0x044e);         // Read SAR04
WritePortUshortAddr,SAR04);               // Restore SAR04
WritePortUchar(ulXrAddr,XR_Index);        // Restore XR_Index

```

4.2.3 How to start video capture

```

// In 55x VGAs, capture counters are not updated with the new off-screen
// address until the next Input Video VSync. So, the first frame of the input
// video is captured at the old address left in the counters when we froze
// the video. This may cause the memory corruption. To avoid this problem we
// need to ignore the data of the first input video frame. We already set the
// acquisition window to NULL during initialization. Now all we have to do is to
// wait for the first couple of input Vsyzns then set the acquisition rectangle
// to proper values. Acquisition rectangle must not be set till video started
// flowing in (bVideoFlowingIn = 1). bVideoFlowingIn flag is set to 0 at
// initialization time.
// So let us perform the first frame ritual.
//
if(!bVideoFlowingIn)
{ // This is first time to start 550 video, see if video is flowing?
  start_time = timeGetTime();           // 1 millisec precision
  while(((timeGetTime() - start_time) < 200))
  { // wait for 200 milisec (33.3 ms for 30Hz video) and VSyncActivity
    WritePortUchar(ulMrAddr, MR_VIN_CTRL_4);
    if(ReadPortUshort(ulMrAddr) & (VIC4_VSYNC << 8))
    {
      bVideoFlowingIn = 1;              // video start flowing
    }
  }
  //
  // Wait for Input VSync is over.
  //
  start_time = timeGetTime();           // 1 millisec precision
  WritePortUchar(ulMrAddr, MR_VIN_CTRL_4);
  while((ReadPortUshort(ulMrAddr) & (VIC4_VSYNC << 8)) &&
        ((timeGetTime() - start_time) < 200));
  //
  // Now wait for next VSync.
  //
  start_time = timeGetTime();           // 1 millisec precision
  WritePortUchar(ulMrAddr, MR_VIN_CTRL_4);
  while(!(ReadPortUshort(ulMrAddr) & (VIC4_VSYNC << 8)) &&
        ((timeGetTime() - start_time) < 200));
  //
  // Restore crop.right
  //
  SetCropRect((LPRECTL)&rCrop);
  break;
}
} // FirstTime VideoFlowingIn
mr03 |= (VIC2_START_GRAB << 8); // unfreeze the video (start capturing)
WritePortUshort(ulMrAddr,mr03);        // write new value
}

```

4.2.4 How to stop video capture

```
//-----
// FreezeVideo() : Stops capturing the incoming video; whatever is in the
//   frame buffer is being displayed.
//
// Enter:
//   none
// Exit :
//   Nothing
//-----
void FreezeVideo()
{
    int    mr03;
    DWORD start_time;

    WritePortUchar(ulMrAddr, MR_VIN_CTRL_2);
    mr03 = ReadPortUshort(ulMrAddr);    // Read current value

    if((mr03 & (VIC2_START_GRAB << 8)))
    { // video is running, h/w is grabbing video, wait for input VSync
        mr03 &= ~(VIC2_START_GRAB << 8);    // turn off the bit to freeze the video
    }
    //
    // Sometimes if Video Input is not coming thru video port (ZV Port
    // Disabled) then we will never get Video VSync (hanging problem). We must
    // time out our wait for VSync. If we do not see VSync within 2 frames
    // of input VSync (80 Milliseconds for 25Hz Video, worst case) we must get
    // out of the waiting loop.
    //
    start_time = timeGetTime();    // 1 millisec precision
    WritePortUchar(ulMrAddr, MR_VIN_CTRL_4);
    while( !(ReadPortUshort(ulMrAddr) & (VIC4_VSYNC << 8)) &&
        ((timeGetTime() - start_time) < 200));

    WritePortUshort(ulMrAddr, mr03);    // freeze the video

    start_time = timeGetTime();    // 1 millisec precision
    WritePortUchar(ulMrAddr, MR_VIN_CTRL_4);
    while((ReadPortUshort(ulMrAddr) & (VIC4_FRM_READY << 8)) &&
        ((timeGetTime() - start_time) < 200));
}
```

4.2.5 How to set input video color format

CHIPS 6555x supports three basic color formats which are YUV4:2:2, RG555 and RGB565. Each format is 16 bit per pixel. The 6555x also allows swapping of the UV positions within a 32 bit dword. The default sequence for YUV4:2:2 is Byte0=Y0, Byte1=U, Byte2=Y1, Byte3=V. The following code shows the input video format selection.

```
//-----
// SetVideoInputFormat() - sets Video Transfer Format bits
//-----
int ChipsVideoOverlay::SetVideoInputFormat(int iVideoFormat)
{
    UINT i;

    WritePortUchar(ulMrAddr, MR_VIN_CTRL_1); // read video display control reg1
    i = ReadPortUshort(ulMrAddr);
    i &= ~(VIC1_FORMAT << 8); // clear format bits (0 is YUV4:2:2)
    switch(iVideoFormat)
    {
    case CMM_FMT_YUV_422:
//      i |= (VIC1_YUV422 << 8); // 0 is YUV 4:2:2
        break;
    case CMM_FMT_RGB_555:
        i |= (VIC1_RGB555 << 8);
        break;
    case CMM_FMT_RGB_565:
        i |= (VIC1_RGB565 << 8);
        break;
    default:
        return;
    }
    WritePortUshort(ulMrAddr, i); // write new format
}
```

4.2.6 How to set interlaced or non-interlaced video input

CHIPS 6555x supports interlaced or non-interlaced video sources. Usually, the NTSC/PAL video sources are interlaced and the hardware MPEG decoder generates non-interlaced video source. Following code selects video input type.

```
void SetVideoInputBits(BOOL interlaced)
{ // interlaced = 1 for interlaced video source
    int mr02;

    WritePortUchar(ulMrAddr, MR_VIN_CTRL_1);
    mr02 = ReadPortUshort(ulMrAddr); // Read current value
    mr02 &= ~(VIC1_NONINTERLACE << 8); // assume interlaced video
    if(interlaced) mr02 |= (VIC1_NONINTERLACE << 8);
    WritePortUshort(ulMrAddr, mr02); // write new value
}
```


4.2.7 How to enable/disable double buffer

CHIPS 6555x supports double buffering for the video capture and playback. Double buffering needs more memory but it minimizes the tearing effect generated by fast changing pictures. We assume that there is enough memory to accommodate both buffers and that the buffer address is programmed in (MR06, MR07, MR08, MR09, MR0A, MR0B). The following code sets/resets double buffering.

```
void SetDoubleBuffer(BOOL double_buffer)
{ // double_buffer = 1 to enable double buffer
  int  mr04, mr20;

  WritePortUchar(ulMrAddr, MR_VIN_CTRL_3);
  mr04 = ReadPortUshort(ulMrAddr); // Read current value
  mr04 &= ~((VIC3_DB_VLOCK+VIC3_ENABLE_DB) << 8); // assume no double buffer
  if(interlaced) mr04 |= ((VIC3_DB_VLOCK+VIC3_ENABLE_DB) << 8);
  WritePortUshort(ulMrAddr, mr04); // write new value
//
// Enable double buffer for video playback which locked with the input VSync.
//
  WritePortUchar(ulMrAddr, MR_VDP_CTRL_3);
  mr20 = ReadPortUshort(ulMrAddr); // Read current value
  mr20 &= ~((VDC3_DB_VLOCK+VDC3_DB_TRIGGER) << 8); // assume no double buffer
  if(interlaced) mr20 |= ((VDC3_DB_VLOCK+VDC3_DB_TRIGGER) << 8);
  WritePortUshort(ulMrAddr, mr20); // write new value
}
```

4.2.8 How to scale input video (before acquiring into frame buffer)

CHIPS 6555x can scale down the video before capturing into the off-screen buffer.

```
//-----
// SetVideoInputScale() : Sets video input scaling factors. Video input scaling
// factor depends on aquisition rectangle and frame buffer rectangle (source
// rectangle).
//
// Enter:
//   wCrop = crop rectangle width
//   hCrp  = crop rectangle height
//   wCap  = Capture buffer width
//   hCap  = Capture buffer height
// Exit :
//   Nothing
//-----
void SetVideoInputScale(int wCrop, int hCrop, int wCap, int hCap)
{
    UINT  mr03, scale_x, scale_y;

    WritePortUchar(ulMrAddr, MR_VIN_CTRL_2);
    mr03 = ReadPortUshort(ulMrAddr);
    mr03 &= ~(VIC2_SCALE_X | VIC2_SCALE_Y) << 8; // assume no scaling

    if(wCrop > wCap)
    { // horizontal input scaling needed
        scale_x = (int)(((DWORD)wCap*VIN_SCALE_X_MAX) / (DWORD)wCrop);
        WritePortUshort(ulMrAddr, (scale_x << 8) | MR_VIN_SCALE_X);
        mr03 |= (VIC2_SCALE_X << 8); // enable input scaling
    }
    if(hCrop > hCap)
    { // vertical input scaling needed
        scale_y = (int)(((DWORD)hCap*VIN_SCALE_Y_MAX) / (DWORD)hCrop);
        WritePortUshort(ulMrAddr, (scale_y << 8) | MR_VIN_SCALE_Y);
        mr03 |= (VIC2_SCALE_Y << 8); // enable input scaling
    }
    WritePortUshort(ulMrAddr, mr03); // set scale factors
}
```

4.2.9 How to crop input video (programming of acquisition rectangle)

Video acquisition rectangle is used to crop the unwanted video input data before the 6555x hardware scales it and grabs it into off-screen buffer. This is also used to crop vertical blank interval data (Closed Caption or Tele Text) from the NTSC video.

```
//-----
// SetCropRect() : Sets cropping rectangle on input video rectangle.
//
//
//      +-----+
//      |         |
//      | CropRect |
//      |         |
//      |         |
//      |         |
//      | Input Video |
//      |         |
//      +-----+
//
// Enter:
//   lpRect Crop Rectangle withing Input Video rectangle (NTSC/PAL dependent)
//   rCrop - local copy of Cropping Rectangle
//   bVideoFlowingIn = 0 if video is not flowing into the port, 1 normally.
// Exit :
//   Nothing
//-----
void SetCropRect(LPRECTL lpRect)
{
    UINT  mr0e,mr0f,mr10,mr11,mr12,mr13,mr14,mr15;
    union WORD16 u;

    if(&rCrop != lpRect) rCrop = *lpRect; // copy crop rectangle into our area
    if(!bVideoFlowingIn) return;
    // Use NULL Rectangle done by static initialization
    u.w = (USHORT)((int)rCrop.left);
    mr0e = ((UINT)u.b[0] << 8) + MR_VIN_AQW_XL_L;
    mr0f = ((UINT)u.b[1] << 8) + MR_VIN_AQW_XL_H;
    u.w = (USHORT)((int)rCrop.right -1);
    mr10 = ((UINT)u.b[0] << 8) + MR_VIN_AQW_XR_L;
    mr11 = ((UINT)u.b[1] << 8) + MR_VIN_AQW_XR_H;
    u.w = (USHORT)((int)rCrop.top);
    mr12 = ((UINT)u.b[0] << 8) + MR_VIN_AQW_YT_L;
    mr13 = ((UINT)u.b[1] << 8) + MR_VIN_AQW_YT_H;
    u.w = (USHORT)((int)rCrop.bottom -1);
    mr14 = ((UINT)u.b[0] << 8) + MR_VIN_AQW_YB_L;
    mr15 = ((UINT)u.b[1] << 8) + MR_VIN_AQW_YB_H;

    WritePortUshort(ulMrAddr,mr0e); // program Left
    WritePortUshort(ulMrAddr,mr0f);
    WritePortUshort(ulMrAddr,mr10); // program right
    WritePortUshort(ulMrAddr,mr11);
    WritePortUshort(ulMrAddr,mr12); // program top
    WritePortUshort(ulMrAddr,mr13);
    WritePortUshort(ulMrAddr,mr14); // program bottom
    WritePortUshort(ulMrAddr,mr15);
}
```

Definition of CHIPSMH.H

```

/*****
* Description: Hardware register definitiona file for 6555x
* Copyright (C) Chips and Technologies, Inc. 1995
*****/

#define WritePortUchar(p,v)      outp((USHORT)p,v)
#define WritePortUshort(p,v)    outpw((USHORT)p,v)
#define WritePortUlong(p,v)     outpd((USHORT)p,v)
#define ReadPortUchar(p)        inp((USHORT)p)
#define ReadPortUshort(p)       inpw((USHORT)p)
#define ReadPortUlong(p)        inpd((USHORT)p)

//-----
// Chips multimedia register description for 6555x registers.
// Any chages here must also be made in CHIPSMH.INC
//-----

#define ADDR_FR      0x03d0      // C&T Flat Panel Register address port
#define DATA_FR     0x03d1      // C&T Flat Panel Register data port
#define ADDR_MR      0x03d2      // C&T Multimedia Register address port
#define DATA_MR     0x03d3      // C&T Multimedia Register data port
#define ADDR_EXTR    0x03d6      // C&T XR Address
#define DATA_EXTR   0x03d7      // C&T XR Data

#define MR_CAPS_REG_1 0x00      // Multimedia capabilities reg 1
#define MR_CAPS_REG_2 0x01      // Multimedia capabilities reg 2
#define MR_VIN_CTRL_1 0x02      // Video Input Control Reg 1
#define MR_VIN_CTRL_2 0x03      // Video Input Control Reg 2
#define MR_VIN_CTRL_3 0x04      // Video Input Control Reg 3
#define MR_VIN_CTRL_4 0x05      // Video Input Control Reg 4(stat reg)
#define MR_VIN_ADDR_1_L 0x06     // Video Input Address Pointer 1 (low)
#define MR_VIN_ADDR_1_M 0x07     // Video Input Address Pointer 1 (mid)
#define MR_VIN_ADDR_1_H 0x08     // Video Input Address Pointer 1 (high)
#define MR_VIN_ADDR_2_L 0x09     // Video Input Address Pointer 2 (low)
#define MR_VIN_ADDR_2_M 0x0A     // Video Input Address Pointer 2 (mid)
#define MR_VIN_ADDR_2_H 0x0B     // Video Input Address Pointer 2 (high)
#define MR_VIN_PITCH_QD 0x0C     // Pitch of Video Input buff in quad
                                // words (8 bytes = 1QD)
#define MR_VIN_AQW_XL_L 0x0E     // Aquisition Window X-Left low
#define MR_VIN_AQW_XL_H 0x0F     // Aquisition Window X-Left high
#define MR_VIN_AQW_XR_L 0x10     // Aquisition Window X-Right low
#define MR_VIN_AQW_XR_H 0x11     // Aquisition Window X-Right high
#define MR_VIN_AQW_YT_L 0x12     // Aquisition Window Y-Top low
#define MR_VIN_AQW_YT_H 0x13     // Aquisition Window Y-Top high
#define MR_VIN_AQW_YB_L 0x14     // Aquisition Window Y-Bottom low
#define MR_VIN_AQW_YB_H 0x15     // Aquisition Window Y-Bottom high
#define MR_VIN_SCALE_X 0x16      // Video Input Horizontal scale factor
#define MR_VIN_SCALE_Y 0x17      // Video Input Vertical scale factor
#define MR_VIN_FRMCOUNT 0x18     // Frame Count for Nth Frame capturing

//-----
// Video Display Registers:
//-----
#define MR_VDP_CTRL_1 0x1E      // Video Display Control Reg 1
#define MR_VDP_CTRL_2 0x1F      // Video Display Control Reg 2
#define MR_VDP_CTRL_3 0x20      // Video Display Control Reg 3
#define MR_VDP_CTRL_4 0x21      // Video Display Control Reg 4 (status)
#define MR_VDP_ADDR_1_L 0x22     // Video Display Addr Pointer 1 (low)
#define MR_VDP_ADDR_1_M 0x23     // Video Display Addr Pointer 1 (mid)
#define MR_VDP_ADDR_1_H 0x24     // Video Display Addr Pointer 1 (high)
#define MR_VDP_ADDR_2_L 0x25     // Video Display Addr Pointer 2 (low)
#define MR_VDP_ADDR_2_M 0x26     // Video Display Addr Pointer 2 (mid)
#define MR_VDP_ADDR_2_H 0x27     // Video Display Addr Pointer 2 (high)

```

```

#define MR_VDP_PITCH_QD 0x28 // Pitch of Video Display Window in
// quad words (8 bytes = 1QD)
#define MR_VDP_WIN_XL_L 0x2A // Display Window X-Left low
#define MR_VDP_WIN_XL_H 0x2B // Display Window X-Left high
#define MR_VDP_WIN_XR_L 0x2C // Display Window X-Right low
#define MR_VDP_WIN_XR_H 0x2D // Display Window X-Right high
#define MR_VDP_WIN_YT_L 0x2E // Display Window Y-Top low
#define MR_VDP_WIN_YT_H 0x2F // Display Window Y-Top high
#define MR_VDP_WIN_YB_L 0x30 // Display Window Y-Bottom low
#define MR_VDP_WIN_YB_H 0x31 // Display Window Y-Bottom high
#define MR_VDP_ZOOM_X 0x32 // Video Display Horizontal Zoom factor
#define MR_VDP_ZOOM_Y 0x33 // Video Display Vertical Zoom factor

//-----
// Color key registers
//-----
#define MR_VDP_CKEY_CTRL 0x3C // Video Color Key Control
#define MR_VDP_CKEY_0 0x3F //sw Graphics Color Key Reg 0 (blue)
#define MR_VDP_CKEY_1 0x3E // Graphics Color Key Reg 1 (green)
#define MR_VDP_CKEY_2 0x3D //sw Graphics Color Key Reg 2 (red)
#define MR_VDP_CKEY_M0 0x42 //sw Graphics Color Key Mask Reg0(blue)
#define MR_VDP_CKEY_M1 0x41 // Graphics Color Key Mask Reg1 (green)
#define MR_VDP_CKEY_M2 0x40 //sw Graphics Color Key Mask Reg2 (red)
#define MR_CRT_SCAN_LO 0x43 // Current CRTC Refresh Scanline Line
// Read Counter lo 8 bits
#define MR_CRT_SCAN_HI 0x44 // Current CRTC Refresh Scanline Line
// Read Counter hi 4 bits

//-----
// Multimedia capabilities register_1 definitions (MR00):
//-----
#define MCAPS_PLAYBACK 0x01 // Play back available
#define MCAPS_CAPTURE 0x02 // Capture available

//-----
// Bit definition of Video Input Control Register1 (MR_VIN_CTRL_1)
//-----
#define VIC1_NONINTERLACE 0x01 // Interlaced video input
#define VIC1_GAMEFORMAT 0x02 // Game format (duplicate field) video
#define VIC1_YUV422 0x00 // Video Input is YUV
#define VIC1_RGB565 0x04 // RGB16 video input (0 is YUV)
#define VIC1_RGB555 0x0C // RGB15 video input
#define VIC1_FORMAT 0x0E // all format bits
#define VIC1_HSYNC_HI 0x10 // H-Sync Polarity : Hi asserted
#define VIC1_VSYNC_HI 0x20 // V-Sync Polarity : Hi asserted
#define VIC1_FLD_DT_INV 0x40 // Field detect polarity inverted
#define VIC1_FLD_DT_LDE 0x80 // Field detect method leading edge

//-----
// Bit definition of Video Input Control Register2 (MR_VIN_CTRL_2)
//-----
#define VIC2_START_GRAB 0x01 // 1:start grab, 0:stop grab
#define VIC2_SINGLE 0x02 // 1:single frame, 0:continuous
#define VIC2_FIELD_GRAB 0x04 // 1:field grab, 0:frame grab
#define VIC2_ODD_FIELD 0x08 // 1:odd field grab, 0:even filed grab
#define VIC2_SCALE_X 0x10 // 1:enable x_scaling, 0:full screen
#define VIC2_SCALE_Y 0x20 // 1:enable y_scaling, 0:full screen
#define VIC2_YSCALE_ES 0x40 // 1:y-scale even spaced, 0:normal
#define VIC2_YSCALE_OW 0x80 // y-scale overwrite, 0:as per prev bit

//-----
// Bit definition of Video Input Control Register3 (MR_VIN_CTRL_3)
//-----
#define VIC3_X_MIRRORED 0x01 // capture direction, 1:right to left,
// 0: left to right

```

```

#define VIC3_Y_FLIPPED 0x02 // capture direction, 1:bottom to top,
                             // 0: top to bottom
#define VIC3_HFILTER 0x04 // 1:enable horizontal filter at input,
                             // 0:no h filter
#define VIC3_DB_VLOCK 0x08 // 1:DoubleBuffer Vsync locked,
                             // 0:DoubleBuffer CPU forced
#define VIC3_ENABLE_DB 0x10 // 1:Enable DoubleBuffer,
                             // 0:No DoubleBuffer
#define VIC3_PTR1_INUSE 0x20 // 1:PTR 1 in use for DoubleBuffer,
                             // 0:PTR 0 in use for DoubleBuffer
#define VIC3_CAPTURE_NF 0x80 // 1:Capture Nth Frame/Field,
                             // 0:Capture single frame

//-----
// Bit definition of Video Input Status Register (MR_VIN_CTRL_4)
//-----
#define VIC4_FRM_READY 0x01 // 1:Frame is ready for grab by CPU
                             // (syncd with VSync)
#define VIC4_VSYNC 0x08 // VSync after polarity correction
                             // (read only)

#define VIC4_PQE_PIXEL 0x10 // 1:Pixel Qualifier as valid pixel
                             // 0:Pixel Qualifier as Blank signal
#define VIC4_PQP_INV 0x20 // 1:Pixel Qualifier polarity inverted,
                             // 0:Pixel Qualifier normal
#define VIC4_SWAP_UV 0x40 // 1:Swap U & V,
                             // 0:UV Normal sequence
#define VIC4_HY_LUV 0x80 // 1:Y on high and UV on low pins(VESA)
                             // 0:UV on high and Y on low pins

//-----
// Bit definition of Video Display Control Register1 (MR_VDP_CTRL_1)
//-----
#define VDC1_X_MIRRORED 0x01 // 1:mirrored (right to left),
                             // 0:normal (left to right)
#define VDC1_Y_FLIPPED 0x02 // 1:Flipped (bottom to top),
                             // 0:normal (top to bottom)
#define VDC1_ZOOM_X 0x04 // 1:enable x_zoom (zoom based on reg),
                             // 0:normal
#define VDC1_ZOOM_Y 0x08 // 1:enable y_zoom (zoom based on reg),
                             // 0:normal
#define VDC1_INTERLACE 0x10 // 1:VGA Mode is interlaced,
                             // 0:non-interlaced mode

//-----
// Bit definition of Video Display Control Register2 (MR_VDP_CTRL_2)
//-----
#define VDC2_YUV422 0x00 // Video Buf is YUV4:2:2
#define VDC2_UV_SWAP 0x01 // Video Buf is YUV4:2:2 with UV Swap
#define VDC2_SIGNED_UV 0x02 // Video Buf is YUV4:2:2 with Signed UV
#define VDC2_YUV422_UVS 0x01 // Video Buf is YUV4:2:2 with UV Swap
#define VDC2_YUV422_SUV 0x02 // Video Buf is YUV4:2:2 with Signed UV
#define VDC2_YUV422_UV 0x03 // Video Buf is YUV4:2:2 Signed UV&Swap
#define VDC2_RGB555 0x09 // Video Buffer is RGB15 (5-5-5)
#define VDC2_RGB565 0x08 // Video Buffer is RGB16 (5-6-5)
#define VDC2_FORMAT 0x1F // All format bits
#define VDC2_H_INTERPOL 0x20 // Enable Horizontal Interpolation
#define VDC2_VI_RUNAVRG 0x40 // Vertical Interpolation is done as
                             // running average method
#define VDC2_V_INTERPOL 0x80 // Enable Vertical Interpolation

//-----
// Bit definition of Video Display Control Register3 (MR_VDP_CTRL_3)
//-----
#define VDC3_DB_TRIGGER 0x08 // Display new pointer on next VSync

```

```

#define      VDC3_DB_CPU_PTR      0x04    // 1:Dbl buf src is buf ptr set by CPU
                                           // 0:Dbl buf src is Input Aqisition's
                                           // last frame
#define      VDC3_DB_PTR2        0x10    // 1:CPU buffer is pointer 2
                                           // 0:CPU buffer is pointer 1
#define      VDC3_DB_VLOVK       0x20    // 1:Double buffer is VSync locked
                                           // 0:Double buffer is unlocked

//-----
// Bit definition of Video Display Status Register4 (MR_VDP_CTRL_4)
//-----
#define      VDC4_DB_PENDING      0x01    // 1:hasn't displayed CPU set buffer
                                           // 0:CPU Set buffer is displayed or in
                                           // process of being displayed
#define      VDC4_DB_USEPTR2      0x02    // 1:PTR2 is being displayed
                                           // 0:PTR1 is being displayed

//-----
// Bit definition of Video Color Key Control Register (MR_VDP_CKEY_CTRL)
//-----
#define      VDC_EV_OVERLAY       0x01    // 1:Enable video overlay
                                           // 0:Display graphics only
#define      VDC_EV_COLOR_KEY     0x02    // 1:Enable video display using clr key
                                           // 0:Color Key Disabled
#define      VDC_EV_XY_RECT       0x04    // 1:Enable video display in Rect Rgn
                                           // 0:Video Display in Rect Rgn disabled
#define      VDC_ENABLE_VAFC      0x08    // 1:Enable external VAFC (like 545)
                                           // for color key only
                                           // 0:Our own video play back
#define      VDC_VAFC_18          0x10    // 1:18 bit external VAFC
                                           // 0:16 bit external VAFC
#define      VDC_BIT_15_KEY       0x40    // 1:in 16BPP modes MSB is routed thru
                                           // Blue0 for color key
                                           // 0:normal color key
#define      VDC_BIT_0_KEY        0x80    // 1:enable blue0 clr key for 16/24BP
                                           // 0:normal color key for 16/24BPP mode
#define      VIN_SCALE_X_MAX      0x100   // max value of x_scale reg (8 bit reg)
#define      VIN_SCALE_Y_MAX      0x100   // max value of y_scale reg (8 bit reg)
// #define      VDP_ZOOM_X_MAX      0x100   // max value of x_zoom reg (ES1 100h)
// #define      VDP_ZOOM_Y_MAX      0x100   // max value of y_zoom reg (ES1 100h)
#define      VDP_ZOOM_X_MAX       0x40    // max value of x_zoom reg (ES0 40h)
#define      VDP_ZOOM_Y_MAX       0x40    // max value of y_zoom reg (ES0 40h)

/*

```



Chips and Technologies, Inc.
2950 Zanker Road
San Jose, California 95134
Phone: 408-434-0600
FAX: 408-894-2080

Title: 6555x HiQVideo™ Series
Programming Examples
Publication No: AN105.2
Stock No.: 020105-002
Revision No: 1.2
Date: 5/13/97