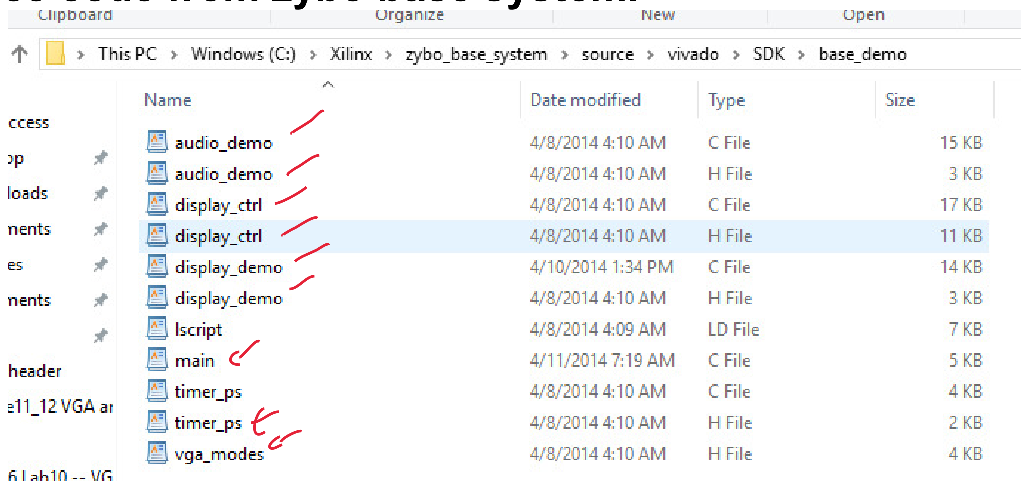
Source code from zybo base system.



Clipboard Organize New Open

This PC > Windows (C:) > Xilinx > zybo_base_system > source > vivado > SDK > base_demo

Name	Date modified	Type	Size
audio_demo	4/8/2014 4:10 AM	C File	15 KB
audio_demo	4/8/2014 4:10 AM	H File	3 KB
display_ctrl	4/8/2014 4:10 AM	C File	17 KB
display_ctrl	4/8/2014 4:10 AM	H File	11 KB
display_demo	4/10/2014 1:34 PM	C File	14 KB
display_demo	4/8/2014 4:10 AM	H File	3 KB
lscript	4/8/2014 4:09 AM	LD File	7 KB
main	4/11/2014 7:19 AM	C File	5 KB
timer_ps	4/8/2014 4:10 AM	C File	4 KB
timer_ps	4/8/2014 4:10 AM	H File	2 KB
vga_modes	4/8/2014 4:10 AM	H File	4 KB

main.c

```
/************************************************************************/
    /*                                                                   */
    /* main.c    --    ZYBO Base System demonstration                    */
    /*                                                                   */
    /*********************************************************************/
    /* Author: Sam Bobrowicz                                             */
    /* Copyright 2014, Digilent Inc.                                     */
    /*********************************************************************/
    /*  Module Description:                                              */
    /*                                                                   */
    /*          This file contains code for running a demonstration of the */
    /*          Video output and audio capabilities of the ZYBO.         */
    /*                                                                   */
    /*********************************************************************/
    /*  Revision History:                                                */
    /*                                                                   */
    /*          2/25/2014(SamB): Created                                 */
    /*                                                                   */
    /*********************************************************************/

    /* ------------------------------------------------------------ */
    /*              Include File Definitions                        */
    /* ------------------------------------------------------------ */

    #include "display_demo.h"
    #include "audio_demo.h"
    #include "timer_ps.h"
    #include "xparameters.h"
    #include "xuartps.h"

    /* ------------------------------------------------------------ */
    /*              Global Variables                                */
    /* ------------------------------------------------------------ */

    /*
     * Display Driver structs
     */
    DisplayCtrl hdmiCtrl;
    DisplayCtrl vgaCtrl;

    /*
     * Framebuffers for each display device
     */
    u32 vgaBuf[DISPLAY_NUM_FRAMES][DISPLAYDEMO_MAX_FRAME];
    u32 hdmiBuf[DISPLAY_NUM_FRAMES][DISPLAYDEMO_MAX_FRAME];
```

Video buffers

page 21, line 84

page 11

(1920*1080

3

u32

unsigned 32 bits

1920x1080 (2K TV)

```c

/*
 * XPAR redefines
 */
#define VGA_BASEADDR XPAR_AXI_DISPCTRL_0_S_AXI_BASEADDR
#define HDMI_BASEADDR XPAR_AXI_DISPCTRL_1_S_AXI_BASEADDR
#define VGA_VDMA_ID XPAR_AXIVDMA_0_DEVICE_ID
#define HDMI_VDMA_ID XPAR_AXIVDMA_1_DEVICE_ID
#define AUDIO_IIC_ID XPAR_XIICPS_0_DEVICE_ID
#define AUDIO_CTRL_BASEADDR XPAR_AXI_I2S_ADI_1_S_AXI_BASEADDR
#define SCU_TIMER_ID XPAR_SCUTIMER_DEVICE_ID
#define UART_BASEADDR XPAR_PS7_UART_1_BASEADDR
#define SW_BASEADDR XPAR_SWS_4BITS_BASEADDR
#define BTN_BASEADDR XPAR_BTNS_4BITS_BASEADDR

void MainDemoPrintMenu();

/* ------------------------------------------------------------ */
/*                Procedure Definitions                         */
/* ------------------------------------------------------------ */

int main(void)
{
   u32 *vgaPtr[DISPLAY_NUM_FRAMES];
```

frame
→ } pointers

```c
   u32 *hdmiPtr[DISPLAY_NUM_FRAMES];
   int i;
   char userInput = 0;

   for (i = 0; i < DISPLAY_NUM_FRAMES; i++)
   {
		vgaPtr[i] = vgaBuf[i];
		hdmiPtr[i] = hdmiBuf[i];
   }

   DisplayDemoInitialize(&vgaCtrl, VGA_VDMA_ID, SCU_TIMER_ID, VGA_BASEADDR, DISPLAY_NOT_HDMI, vgaPtr);
   DisplayDemoInitialize(&hdmiCtrl, HDMI_VDMA_ID, SCU_TIMER_ID, HDMI_BASEADDR, DISPLAY_HDMI, hdmiPtr);
   AudioInitialize(SCU_TIMER_ID, AUDIO_IIC_ID, AUDIO_CTRL_BASEADDR);
   TimerInitialize(SCU_TIMER_ID);

   /* Flush UART FIFO */
   while (XUartPs_IsReceiveData(UART_BASEADDR))
   {
		XUartPs_ReadReg(UART_BASEADDR, XUARTPS_FIFO_OFFSET);
   }

   while (userInput != 'q')
   {
		MainDemoPrintMenu();

		/* Wait for data on UART */
		while (!XUartPs_IsReceiveData(UART_BASEADDR))
		{}

		/* Store the first character in the UART recieve FIFO and echo it */
		userInput = XUartPs_ReadReg(UART_BASEADDR, XUARTPS_FIFO_OFFSET);
		xil_printf("%c", userInput);

		switch (userInput)
		{
		case '1':
			AudioRunDemo(AUDIO_CTRL_BASEADDR, UART_BASEADDR, SW_BASEADDR, BTN_BASEADDR);
			break;
		case '2':
```

```
                        DisplayDemoRun(&vgaCtrl, UART_BASEADDR);
                        break;
                case '3':
                        DisplayDemoRun(&hdmiCtrl, UART_BASEADDR);
                        break;
                case 'q':
                        break;
                default :
                        xil_printf("\n\rInvalid Selection");
                        TimerDelay(500000);
                }
        }

        return 0;
}

void MainDemoPrintMenu()
{
        xil_printf("\x1B[H"); //Set cursor to top left of terminal
        xil_printf("\x1B[2J"); //Clear terminal
        xil_printf("**************************************************\n\r");
        xil_printf("**************************************************\n\r");
        xil_printf("*         ZYBO Base System User Demo             *\n\r");
        xil_printf("**************************************************\n\r");
        xil_printf("**************************************************\n\r");
        xil_printf("\n\r");
        xil_printf("1 - Audio Demo\n\r");
        xil_printf("2 - VGA output demo\n\r");
        xil_printf("3 - HDMI output demo\n\r");
        xil_printf("q - Quit\n\r");
        xil_printf("\n\r");
        xil_printf("Select a demo to run:");
}
/* end main.c*/
```

display_demo.c

```
/************************************************************************/
	/*                                                                      */
	/* display_demo.c  --     ZYBO Display demonstration                   */
	/*                                                                      */
	/************************************************************************/
	/* Author: Sam Bobrowicz                                                */
	/* Copyright 2014, Digilent Inc.                                        */
	/************************************************************************/
	/*  Module Description:                                                 */
	/*                                                                      */
	/*		This file contains code for running a demonstration of the      */
	/*		Video output capabilities on the ZYBO. It is a good example of   */
	/*		how to properly use the display_ctrl driver.                     */
	/*                                                                      */
	/*		This module contains code from the Xilinx Demo titled            */
	/*		" xiicps_polled_master_example.c"                                */
	/*                                                                      */
	/************************************************************************/
	/*  Revision History:                                                   */
	/*                                                                      */
	/*		2/20/2014(SamB): Created                                         */
	/*                                                                      */
	/************************************************************************/

	/* ------------------------------------------------------------ */
	/*		Include File Definitions                         */
	/* ------------------------------------------------------------ */

	#include "display_demo.h"
	#include <stdio.h>
	#include "xuartps.h"
	#include "math.h"
	#include <ctype.h>
	#include <stdlib.h>
	#include "xil_types.h"
	#include "xil_cache.h"
	#include "timer_ps.h"
	#include "display_ctrl.h"

	/* ------------------------------------------------------------ */
	/*		Procedure Definitions                            */
	/* ------------------------------------------------------------ */

	int DisplayDemoInitialize(DisplayCtrl *dispPtr, u16 vdmaId, u16 timerId, u32 dispCtrlAddr, int fHdmi, u32
	*framePtr[DISPLAY_NUM_FRAMES])
	{
	   int Status;

	   TimerInitialize(timerId);

	   Status = DisplayInitialize(dispPtr, vdmaId, dispCtrlAddr, fHdmi, framePtr, DISPLAYDEMO_STRIDE);
	   if (Status != XST_SUCCESS)
	   {
			xil_printf("Display Ctrl initialization failed during demo initialization%d\r\n", Status);
			return XST_FAILURE;
	   }

	   Status = DisplayStart(dispPtr);
	   if (Status != XST_SUCCESS)
	   {
			xil_printf("Couldn't start display during demo initialization%d\r\n", Status);
			return XST_FAILURE;
	   }

    DisplayDemoPrintTest(dispPtr->framePtr[dispPtr->curFrame], dispPtr->vMode.width, dispPtr->vMode.height, dispPtr->stride, DISPLAYDEMO_PATTERN_1);

    return XST_SUCCESS;
}

void DisplayDemoPrintMenu(DisplayCtrl *dispPtr)
{
    xil_printf("\x1B[H"); //Set cursor to top left of terminal
    xil_printf("\x1B[2J"); //Clear terminal
    xil_printf("**************************************************\n\r");
    xil_printf("*             ZYBO Display User Demo             *\n\r");
    xil_printf("**************************************************\n\r");
    xil_printf("*Port: %42s*\n\r", (dispPtr->fHdmi == DISPLAY_HDMI) ? "HDMI" : "VGA");
    xil_printf("*Current Resolution: %28s*\n\r", dispPtr->vMode.label);
    printf("*Pixel Clock Freq. (MHz): %23.3f*\n\r", (dispPtr->fHdmi == DISPLAY_HDMI) ? (dispPtr->pxlFreq / 5.0) : (dispPtr->pxlFreq));
    xil_printf("*Current Frame Index: %27d*\n\r", dispPtr->curFrame);
    xil_printf("**************************************************\n\r");
    xil_printf("\n\r");
    xil_printf("1 - Change Resolution\n\r");
    xil_printf("2 - Change Frame\n\r");
    xil_printf("3 - Print Blended Test Pattern to current Frame\n\r");
    xil_printf("4 - Print Color Bar Test Pattern to current Frame\n\r");
    xil_printf("5 - Invert Current Frame colors\n\r");
    xil_printf("6 - Invert Current Frame colors seamlessly*\n\r");
    xil_printf("q - Quit\n\r");
    xil_printf("\n\r");
    xil_printf("*Note that option 6 causes the current frame index to be \n\r");
    xil_printf(" incremented. This is because the inverted frame is drawn\n\r");
    xil_printf(" to an inactive frame. After the drawing is complete, this\n\r");
    xil_printf(" frame is then set to be the active frame. This demonstrates\n\r");
    xil_printf(" how to properly update what is being displayed without image\n\r");
    xil_printf(" tearing. Options 3-5 all draw to the currently active frame,\n\r");
    xil_printf(" which is why not all pixels appear to be updated at once.\n\r");
    xil_printf("\n\r");
    xil_printf("Enter a selection:");
}

int DisplayDemoRun(DisplayCtrl *dispPtr, u32 uartAddr)
{
    char userInput = 0;
    int nextFrame = 0;

    /* Flush UART FIFO */
    while (XUartPs_IsReceiveData(uartAddr))
    {
        XUartPs_ReadReg(uartAddr, XUARTPS_FIFO_OFFSET);
    }

    while (userInput != 'q')
    {
        DisplayDemoPrintMenu(dispPtr);

        /* Wait for data on UART */
        while (!XUartPs_IsReceiveData(uartAddr))
        {}

        /* Store the first character in the UART recieve FIFO and echo it */
        userInput = XUartPs_ReadReg(uartAddr, XUARTPS_FIFO_OFFSET);
        xil_printf("%c", userInput);

        switch (userInput)
        {
        case '1':
            DisplayDemoChangeRes(dispPtr, uartAddr);
            break;
        case '2':
            nextFrame = dispPtr->curFrame + 1;
            if (nextFrame >= DISPLAY_NUM_FRAMES)
            {
                nextFrame = 0;
            }
            DisplayChangeFrame(dispPtr, nextFrame);
            break;
        case '3':
            DisplayDemoPrintTest(dispPtr->framePtr[dispPtr->curFrame], dispPtr->vMode.width, dispPtr->vMode.height, dispPtr->stride, DISPLAYDEMO_PATTERN_0);
            break;
        case '4':
            DisplayDemoPrintTest(dispPtr->framePtr[dispPtr->curFrame], dispPtr->vMode.width, dispPtr->vMode.height, dispPtr->stride, DISPLAYDEMO_PATTERN_1);
            break;
        case '5':
            DisplayDemoInvertFrame(dispPtr->framePtr[dispPtr->curFrame], dispPtr->framePtr[dispPtr->curFrame], dispPtr->vMode.width, dispPtr->vMode.height, dispPtr->stride);
            break;
        case '6':
            nextFrame = dispPtr->curFrame + 1;
            if (nextFrame >= DISPLAY_NUM_FRAMES)
            {
                nextFrame = 0;
            }
            DisplayDemoInvertFrame(dispPtr->framePtr[dispPtr->curFrame], dispPtr->framePtr[nextFrame], dispPtr->vMode.width, dispPtr->vMode.height, dispPtr->stride);
            DisplayChangeFrame(dispPtr, nextFrame);
            break;
        case 'q':
            break;
        default :
            xil_printf("\n\rInvalid Selection");
            TimerDelay(500000);
        }
    }

    return XST_SUCCESS;
}

void DisplayDemoChangeRes(DisplayCtrl *dispPtr, u32 uartAddr)
{
    char userInput = 0;
    int fResSet = 0;
    int status;

    /* Flush UART FIFO */
    while (XUartPs_IsReceiveData(uartAddr))
    {
        XUartPs_ReadReg(uartAddr, XUARTPS_FIFO_OFFSET);
    }

    while (!fResSet)
    {
        DisplayDemoCRMenu(dispPtr);

        /* Wait for data on UART */
        while (!XUartPs_IsReceiveData(uartAddr))
        {}

        /* Store the first character in the UART recieve FIFO and echo it */
        userInput = XUartPs_ReadReg(uartAddr, XUARTPS_FIFO_OFFSET);
        xil_printf("%c", userInput);
        status = XST_SUCCESS;
        switch (userInput)
        {
        case '1':
            status = DisplayStop(dispPtr);
            DisplaySetMode(dispPtr, &VMODE_640x480);
            DisplayStart(dispPtr);
            fResSet = 1;
            break;
        case '2':
            status = DisplayStop(dispPtr);
            DisplaySetMode(dispPtr, &VMODE_800x600);
            DisplayStart(dispPtr);
            fResSet = 1;
            break;
        case '3':
            status = DisplayStop(dispPtr);
            DisplaySetMode(dispPtr, &VMODE_1280x720);
            DisplayStart(dispPtr);
            fResSet = 1;
            break;
        case '4':
            status = DisplayStop(dispPtr);
            DisplaySetMode(dispPtr, &VMODE_1280x1024);
            DisplayStart(dispPtr);
            fResSet = 1;
            break;
        case '5':
            status = DisplayStop(dispPtr);
            DisplaySetMode(dispPtr, &VMODE_1920x1080);
            DisplayStart(dispPtr);
            fResSet = 1;
            break;
        case 'q':
            fResSet = 1;
            break;
        default :
            xil_printf("\n\rInvalid Selection");
            TimerDelay(500000);
        }
        if (status == XST_DMA_ERROR)
        {
            xil_printf("\n\rWARNING: AXI VDMA Error detected and cleared\n\r");
        }
    }
}

void DisplayDemoCRMenu(DisplayCtrl *dispPtr)
{
    xil_printf("\x1B[H"); //Set cursor to top left of terminal
    xil_printf("\x1B[2J"); //Clear terminal
    xil_printf("***************************************************\n\r");
    xil_printf("*           ZYBO Display User Demo                *\n\r");
    xil_printf("***************************************************\n\r");
    xil_printf("*Port: %42s*\n\r", (dispPtr->fHdmi == DISPLAY_HDMI) ? "HDMI" : "VGA");
    xil_printf("*Current Resolution: %28s*\n\r", dispPtr->vMode.label);
    printf("*Pixel Clock Freq. (MHz): %23.3f*\n\r", (dispPtr->fHdmi == DISPLAY_HDMI) ? (dispPtr->pxlFreq / 5.0) : (dispPtr->pxlFreq));
    xil_printf("***************************************************\n\r");
    xil_printf("\n\r");
    xil_printf("1 - %s\n\r", VMODE_640x480.label);
    xil_printf("2 - %s\n\r", VMODE_800x600.label);
    xil_printf("3 - %s\n\r", VMODE_1280x720.label);
    xil_printf("4 - %s\n\r", VMODE_1280x1024.label);
    xil_printf("5 - %s\n\r", VMODE_1920x1080.label);
    xil_printf("q - Quit (don't change resolution)\n\r");
    xil_printf("\n\r");
```

```
   xil_printf("Select a new resolution:");
}

void DisplayDemoInvertFrame(u32 *srcFrame, u32 *destFrame, u32 width, u32 height, u32 stride)
{
   u32 xcoi, ycoi;
   u32 lineStart = 0;
   for(ycoi = 0; ycoi < height; ycoi++)
   {
		for(xcoi = 0; xcoi < width; xcoi++)
		{
			destFrame[xcoi + lineStart] = ~srcFrame[xcoi + lineStart];
		}
		lineStart += stride / 4; /*The stride is in bytes, so it needs to be divided by four to get the u32 address*/
   }
   /* Flush the framebuffer memory range to ensure changes are written to the
    * actual memory, and therefore accessible by the VDMA.   */
   Xil_DCacheFlushRange((unsigned int) destFrame, DISPLAYDEMO_MAX_FRAME * 4);
}


void DisplayDemoPrintTest(u32 *frame, u32 width, u32 height, u32 stride, int pattern)
{
   u32 xcoi, ycoi;
   u32 iPixelAddr;
   u32 wStride;
   u32 wRed, wBlue, wGreen, wColor;
   u32 wCurrentInt;
   double fRed, fBlue, fGreen, fColor;
   u32 xLeft, xMid, xRight, xInt;
   u32 yMid, yInt;
   double xInc, yInc;

   switch (pattern)
   {
   case DISPLAYDEMO_PATTERN_0:

		wStride = stride / 4; /* Find the stride in 32-bit words */

		xInt = width / 4; //Four intervals, each with width/4 pixels
		xLeft = xInt;
		xMid = xInt * 2;
		xRight = xInt * 3;
		xInc = 256.0 / ((double) xInt); //256 color intensities are cycled through per interval (overflow must be caught 
when color=256.0)

		yInt = height / 2; //Two intervals, each with width/2 lines
		yMid = yInt;
		yInc = 256.0 / ((double) yInt); //256 color intensities are cycled through per interval (overflow must be caught 
when color=256.0)

		fBlue = 0.0;
		fRed = 256.0;
		for(xcoi = 0; xcoi < width; xcoi++)
		{
			/*
			 * Convert color intensities to integers < 256, and trim values >=256
			 */
			wRed = (fRed >= 256.0) ? 255 : ((u32) fRed);
			wBlue = (fBlue >= 256.0) ? 255 : ((u32) fBlue);

			wColor = (wRed << BIT_DISPLAY_RED) | (wBlue << BIT_DISPLAY_BLUE);
			iPixelAddr = xcoi;
			fGreen = 0.0;
			for(ycoi = 0; ycoi < height; ycoi++)
			{
```

```
                wGreen = (fGreen >= 256.0) ? 255 : ((u32) fGreen);
                frame[iPixelAddr] = wColor | (wGreen << BIT_DISPLAY_GREEN);
                if (ycoi < yMid)
                {
                    fGreen += yInc;
                }
                else
                {
                    fGreen -= yInc;
                }

        /* This pattern is printed one vertical line at a time, so the address must be incremented
         * by the stride instead of just 1.              */
                iPixelAddr += wStride;
            }

            if (xcoi < xLeft)
            {
                fBlue = 0.0;
                fRed -= xInc;
            }
            else if (xcoi < xMid)
            {
                fBlue += xInc;
                fRed += xInc;
            }
            else if (xcoi < xRight)
            {
                fBlue -= xInc;
                fRed -= xInc;
            }
            else
            {
                fBlue += xInc;
                fRed = 0;
            }
        }
        /*Flush the framebuffer memory range to ensure changes are written to the
         * actual memory, and therefore accessible by the VDMA. */
        Xil_DCacheFlushRange((unsigned int) frame, DISPLAYDEMO_MAX_FRAME * 4);
        break;
    case DISPLAYDEMO_PATTERN_1:
        wStride = stride / 4; /* Find the stride in 32-bit words */

        xInt = width / 7; //Seven intervals, each with width/7 pixels
        xInc = 256.0 / ((double) xInt); //256 color intensities per interval. Notice that overflow is handled for this pattern.

        fColor = 0.0;
        wCurrentInt = 1;
        for(xcoi = 0; xcoi < width; xcoi++)
        {
            if (wCurrentInt & 0b001)
                fRed = fColor;
            else
                fRed = 0.0;

            if (wCurrentInt & 0b010)
                fBlue = fColor;
            else
                fBlue = 0.0;

            if (wCurrentInt & 0b100)
                fGreen = fColor;
            else
                fGreen = 0.0;

```

```
			/* Just draw white in the last partial interval (when width is not divisible by 7)	*/
				if (wCurrentInt > 7)
				{
					wColor = 0x00FFFFFF;
				}
				else
				{
					wColor = ((u32) fRed << BIT_DISPLAY_RED) | ((u32) fBlue << BIT_DISPLAY_BLUE) | (
	(u32) fGreen << BIT_DISPLAY_GREEN);
				}
				iPixelAddr = xcoi;

				for(ycoi = 0; ycoi < height; ycoi++)
				{
					frame[iPixelAddr] = wColor;
			/* This pattern is printed one vertical line at a time, so the address must be incremented
			 by the stride instead of just 1.	*/
					iPixelAddr += wStride;
				}

				fColor += xInc;
				if (fColor >= 256.0)
				{
					fColor = 0.0;
					wCurrentInt++;
				}
			}
			/* Flush the framebuffer memory range to ensure changes are written to the
			 * actual memory, and therefore accessible by the VDMA. */
			Xil_DCacheFlushRange((unsigned int) frame, DISPLAYDEMO_MAX_FRAME * 4);
			break;
	default :
			xil_printf("Error: invalid pattern passed to DisplayDemoPrintTest");
	}
}
/*end display_demo.c */
```

display_demo.h

```
/************************************************************************/
	/*                                                                      */
	/* display_demo.h  --         ZYBO Display demonstration                */
	/*                                                                      */
	/************************************************************************/
	/* Author: Sam Bobrowicz                                                */
	/* Copyright 2014, Digilent Inc.                                        */
	/************************************************************************/
	/*  Module Description:                                                 */
	/*                                                                      */
	/*              This file contains code for running a demonstration of the */
	/*              Video output capabilities on the ZYBO. It is a good example of */
	/*              how to properly use the display_ctrl driver.            */
	/*                                                                      */
	/*              This module contains code from the Xilinx Demo titled   */
	/*              " xiicps_polled_master_example.c"                       */
	/*                                                                      */
	/************************************************************************/
	/*  Revision History:                                                   */
	/*                                                                      */
	/*              2/20/2014(SamB): Created                                */
	/*                                                                      */
	/************************************************************************/

	#ifndef DISPLAY_DEMO_H_
	#define DISPLAY_DEMO_H_

	/* ------------------------------------------------------------ */
	/*              Include File Definitions                        */
	/* ------------------------------------------------------------ */

	#include "xil_types.h"
	#include "display_ctrl.h"

	/* ------------------------------------------------------------ */
	/*              Miscellaneous Declarations                      */
	/* ------------------------------------------------------------ */

	#define DISPLAYDEMO_PATTERN_0 0
	#define DISPLAYDEMO_PATTERN_1 1

	#define DISPLAYDEMO_MAX_FRAME (1920*1080)
	#define DISPLAYDEMO_STRIDE (1920 * 4)

	/* ------------------------------------------------------------ */
	/*              Procedure Declarations                          */
	/* ------------------------------------------------------------ */

	int DisplayDemoInitialize(DisplayCtrl *dispPtr, u16 vdmaId, u16 timerId, u32 dispCtrlAddr, int fHdmi, u32 *framePtr[DISPLAY_NUM_FRAMES]);
	void DisplayDemoPrintMenu(DisplayCtrl *dispPtr);
	int DisplayDemoRun(DisplayCtrl *dispPtr, u32 uartAddr);
	void DisplayDemoChangeRes(DisplayCtrl *dispPtr, u32 uartAddr);
	void DisplayDemoCRMenu(DisplayCtrl *dispPtr);
	void DisplayDemoInvertFrame(u32 *srcFrame, u32 *destFrame, u32 width, u32 height, u32 stride);
	void DisplayDemoPrintTest(u32 *frame, u32 width, u32 height, u32 stride, int pattern);

	/* ------------------------------------------------------------ */
	/************************************************************************/
	#endif /* DISPLAY_DEMO_H_ */
```

display_ctrl.c

```
/************************************************************************/
	/*									*/
	/* display_ctrl.c	--	Digilent Display Controller Driver		*/
	/*									*/
	/************************************************************************/
	/* Author: Sam Bobrowicz							*/
	/* Copyright 2014, Digilent Inc.						*/
	/************************************************************************/
	/*  Module Description:							*/
	/*									*/
	/*		This module provides an easy to use API for controlling the	*/
	/*		Digilent display controller core (axi_dispctrl). It supports	*/
	/*		run-time resolution setting and seamless framebuffer-swapping	*/
	/*		for tear-free animation.					*/
	/*									*/
	/*		To use this driver, you must have an axi_dispctrl and axi_vdma	*/
	/*		core present in your system. For information on how to properly */
	/*		configure these cores within your design, refer to the		*/
	/*		axi_dispctrl data sheet accessible from Vivado and XPS.		*/
	/*									*/
	/*		The following steps should be followed to use this driver:	*/
	/*		1) Create a DisplayCtrl object and pass a pointer to it to	*/
	/*		   DisplayInitialize.						*/
	/*		2) Call DisplaySetMode to set the desired mode			*/
	/*		3) Call DisplayStart to begin outputting data to the display	*/
	/*		4) To create a seamless animation, draw the next image to a	*/
	/*		   framebuffer currently not being displayed. Then call		*/
	/*		   DisplayChangeFrame to begin displaying that frame.		*/
	/*		   Repeat as needed, only ever modifying inactive frames.	*/
	/*		5) To change the resolution, call DisplaySetMode, followed by	*/
	/*		   DisplayStart again.						*/
	/*									*/
	/*		This module contains code from the Xilinx Demo titled		*/
	/*		"xiicps_polled_master_example.c." Xilinx XAPP888 was also	*/
	/*		referenced for information on reconfiguring the MMCM or PLL.	*/
	/*		Note that functions beginning with "DisplayClk" are used	*/
	/*		internally for this purpose and should not need to be called	*/
	/*		externally.							*/
	/************************************************************************/
	/*  Revision History:							*/
	/*									*/
	/*		2/20/2014(SamB): Created					*/
	/*									*/
	/************************************************************************/
	/*
	 * TODO: Functionality needs to be added to take advantage of the MMCM's fractional
	 * divide. This will allow a far greater number of frequencies to be synthesized. */
	/* ------------------------------------------------------------ */
	/*				Include File Definitions	*/
	/* ------------------------------------------------------------ */
	#include "display_ctrl.h"
	#include "xil_io.h"
	#include "math.h"
	#include "xil_types.h"
	#include "vga_modes.h"
	#include "xaxivdma.h"
	#include <stdio.h>
	/* ------------------------------------------------------------ */
	/*			Procedure Definitions			*/
	/* ------------------------------------------------------------ */
	/***		DisplayStop(DisplayCtrl *dispPtr)
	**
	**		Parameters:
	**		dispPtr - Pointer to the initialized DisplayCtrl struct
```

```
**
**		Return Value: int
**			XST_SUCCESS if successful.
**			XST_DMA_ERROR if an error was detected on the DMA channel. The
**			Display is still successfully stopped, and the error is
**			cleared so that subsequent DisplayStart calls will be
**			successful. This typically indicates insufficient bandwidth
**			on the AXI Memory-Map Interconnect (VDMA<->DDR)
**
**		Description:
**			Halts output to the display
*/
int DisplayStop(DisplayCtrl *dispPtr)
{
   /*		* If already stopped, do nothing		*/
   if (dispPtr->state == DISPLAY_STOPPED)
   {
		return XST_SUCCESS;
   }

   /*
    * Disable the disp_ctrl core, and wait for the current frame to finish (the core cannot stop  mid-frame)
    */
   Xil_Out32(dispPtr->dispCtrlAddr + OFST_DISPLAY_CTRL, 0);
   while (Xil_In32(dispPtr->dispCtrlAddr + OFST_DISPLAY_STATUS) & (1 << BIT_DISPLAY_RUNNING));

   /*		* Stop the VDMA core		*/
   XAxiVdma_DmaStop(&(dispPtr->vdma), XAXIVDMA_READ);
   while(XAxiVdma_IsBusy(&(dispPtr->vdma), XAXIVDMA_READ));

   /*		* Update Struct state		*/
   dispPtr->pxlFreq = 0;
   dispPtr->state = DISPLAY_STOPPED;

   if (XAxiVdma_GetDmaChannelErrors(&(dispPtr->vdma), XAXIVDMA_READ))
   {
		XAxiVdma_ClearDmaChannelErrors(&(dispPtr->vdma), XAXIVDMA_READ, 0xFFFFFFFF);
		return XST_DMA_ERROR;
   }

   return XST_SUCCESS;
}
/* ------------------------------------------------------------ */
/***		DisplayStart(DisplayCtrl *dispPtr)
**
**		Parameters:
**			dispPtr - Pointer to the initialized DisplayCtrl struct
**		Return Value: int
**			XST_SUCCESS if successful, XST_FAILURE otherwise
**		Errors:
**		Description:
**			Starts the display.
*/
int DisplayStart(DisplayCtrl *dispPtr)
{
   int Status;
   u32 vgaReg[5];
   ClkConfig clkReg;
   ClkMode clkMode;
   int i;
   double pxlClkFreq;

   /*
    * If already started, do nothing
    */
   if (dispPtr->state == DISPLAY_RUNNING)
```

```
	{
		return XST_SUCCESS;
	}

	/* Configure the VDMA to access a frame with the same dimensions as the current mode	*/
	dispPtr->vdmaConfig.VertSizeInput = dispPtr->vMode.height;
	dispPtr->vdmaConfig.HoriSizeInput = (dispPtr->vMode.width) * 4;
	dispPtr->vdmaConfig.FixedFrameStoreAddr = dispPtr->curFrame;
	/*
	 *Also reset the stride and address values, in case the user manually changed them
	 */
	dispPtr->vdmaConfig.Stride = dispPtr->stride;
	for (i = 0; i < DISPLAY_NUM_FRAMES; i++)
	{
		dispPtr->vdmaConfig.FrameStoreStartAddr[i] = (u32)  dispPtr->framePtr[i];
	}

	/*
	 * Perform the VDMA driver calls required to start a transfer. Note that no data is actually
	 * transferred until the disp_ctrl core signals the VDMA core by pulsing fsync.
	 */
	Status = XAxiVdma_DmaConfig(&(dispPtr->vdma), XAXIVDMA_READ, &(dispPtr->vdmaConfig));
	if (Status != XST_SUCCESS)
	{
		xil_printf("Read channel config failed %d\r\n", Status);
		return XST_FAILURE;
	}
	Status = XAxiVdma_DmaSetBufferAddr(&(dispPtr->vdma), XAXIVDMA_READ, dispPtr-
>vdmaConfig.FrameStoreStartAddr);
	if (Status != XST_SUCCESS)
	{
		xil_printf("Read channel set buffer address failed %d\r\n", Status);
		return XST_FAILURE;
	}
	Status = XAxiVdma_DmaStart(&(dispPtr->vdma), XAXIVDMA_READ);
	if (Status != XST_SUCCESS)
	{
		xil_printf("Start read transfer failed %d\r\n", Status);
		return XST_FAILURE;
	}
	Status = XAxiVdma_StartParking(&(dispPtr->vdma), dispPtr->curFrame, XAXIVDMA_READ);
	if (Status != XST_SUCCESS)
	{
		xil_printf("Unable to park the channel %d\r\n", Status);
		return XST_FAILURE;
	}

	/* Configure the disp_ctrl core with the display mode timing parameters */
	vgaReg[0] = (dispPtr->vMode.width << 16) | (dispPtr->vMode.height);
	vgaReg[1] = (dispPtr->vMode.hps << 16) | (dispPtr->vMode.hpe);
	vgaReg[2] = (dispPtr->vMode.hpol << 16) | (dispPtr->vMode.hmax);
	vgaReg[3] = (dispPtr->vMode.vps << 16) | (dispPtr->vMode.vpe);
	vgaReg[4] = (dispPtr->vMode.vpol << 16) | (dispPtr->vMode.vmax);
	for (i = 0; i < 5; i++)
	{
		Xil_Out32(dispPtr->dispCtrlAddr + OFST_DISPLAY_VIDEO_START + (i * 4), vgaReg[i]);
	}

	/* Calculate the PLL divider parameters based on the required pixel clock frequency	*/
	if (dispPtr->fHdmi == DISPLAY_HDMI)
	{
		pxlClkFreq = dispPtr->vMode.freq * 5;
	}
	else
	{
		pxlClkFreq = dispPtr->vMode.freq;
	}
	DisplayClkFindParams(pxlClkFreq, &clkMode);

	/*
	 * Store the obtained frequency to pxlFreq. It is possible that the PLL was not able to
	 * exactly generate the desired pixel clock, so this may differ from vMode.freq.		*/
	dispPtr->pxlFreq = clkMode.freq;

	/*
	 * Write to the PLL dynamic configuration registers to configure it with the calculated parameters.
	 */
	if (!DisplayClkFindReg(&clkReg, &clkMode))
	{
		xil_printf("Error calculating CLK register values\n\r");
		return XST_FAILURE;
	}
	DisplayClkWriteReg(&clkReg, dispPtr->dispCtrlAddr);

	/*
	 * Enable the disp_ctrl core, which will signal the VDMA to begin transferring data
	 */
	Xil_Out32(dispPtr->dispCtrlAddr + OFST_DISPLAY_CTRL, (1 << BIT_DISPLAY_START));

	dispPtr->state = DISPLAY_RUNNING;

	return XST_SUCCESS;
}

/* ------------------------------------------------------------ */
/***	DisplayInitialize(DisplayCtrl *dispPtr, u16 vdmaId, u32 dispCtrlAddr, int fHdmi, u32 *framePtr[DISPLAY_NUM_FRAMES], u32 stride)
**	Parameters:
**		dispPtr - Pointer to the struct that will be initialized
**		vdmaId - DEVICE ID of the attached VDMA core
**		dispCtrlAddr - BASE ADDRESS of the axi_dispctrl core
**		fHdmi - flag indicating if the C_USE_BUFR_DIV5 parameter is set for the axi_dispctrl core.
**		Use DISPLAY_HDMI if it is set, otherwise use DISPLAY_NOT_HDMI
**	framePtr - array of pointers to the framebuffers. The framebuffers must be instantiated above this driver
**stride - line stride of the framebuffers. This is the number of bytes between the start of one line and the start of another.
**
**	Return Value: int
**		XST_SUCCESS if successful, XST_FAILURE otherwise
**	Errors:
**	Description:
**		Initializes the driver struct for use.
*/
int DisplayInitialize(DisplayCtrl *dispPtr, u16 vdmaId, u32 dispCtrlAddr, int fHdmi, u32 *framePtr[DISPLAY_NUM_FRAMES], u32 stride)
{
	int Status;
	int i;
	XAxiVdma_Config *Config;

	/*
	 * Initialize all the fields in the DisplayCtrl struct
	 */
	dispPtr->curFrame = 0;
	dispPtr->dispCtrlAddr = dispCtrlAddr;
	dispPtr->fHdmi = fHdmi;
	for (i = 0; i < DISPLAY_NUM_FRAMES; i++)
	{
		dispPtr->framePtr[i] = framePtr[i];
	}
	dispPtr->pxlFreq = 0;
	dispPtr->state = DISPLAY_STOPPED;
	dispPtr->stride = stride;
	dispPtr->vMode = VMODE_640x480;

	/*		* Initialize VDMA driver	*/
	Config = XAxiVdma_LookupConfig(vdmaId);
	if (!Config)
	{
		xil_printf("No video DMA found for ID %d\r\n", vdmaId);
		return XST_FAILURE;
	}
	Status = XAxiVdma_CfgInitialize(&(dispPtr->vdma), Config, Config->BaseAddress);
	if (Status != XST_SUCCESS)
	{
		xil_printf("Configuration Initialization failed %d\r\n", Status);
		return XST_FAILURE;
	}

	Status = XAxiVdma_SetFrmStore(&(dispPtr->vdma), DISPLAY_NUM_FRAMES, XAXIVDMA_READ);
	if (Status != XST_SUCCESS)
	{
		xil_printf("Setting Frame Store Number Failed in Read Channel %d\r\n", Status);
		return XST_FAILURE;
	}

	/*		* Initialize the VDMA Read configuration struct  */
	dispPtr->vdmaConfig.FrameDelay = 0;
	dispPtr->vdmaConfig.EnableCircularBuf = 1;
	dispPtr->vdmaConfig.EnableSync = 0;
	dispPtr->vdmaConfig.PointNum = 0;
	dispPtr->vdmaConfig.EnableFrameCounter = 0;

	return XST_SUCCESS;
}
/* ------------------------------------------------------------ */
/***		DisplaySetMode(DisplayCtrl *dispPtr, const VideoMode *newMode)
**		Parameters:
**			dispPtr - Pointer to the initialized DisplayCtrl struct
**			newMode - The VideoMode struct describing the new mode.
**		Return Value: int
**			XST_SUCCESS if successful, XST_FAILURE otherwise
**		Errors:
**		Description:
**			Changes the resolution being output to the display. If the display
**			is currently started, it is automatically stopped (DisplayStart must
**			be called again).
*/
int DisplaySetMode(DisplayCtrl *dispPtr, const VideoMode *newMode)
{
	int Status;

	/*
	 * If currently running, stop
	 */
	if (dispPtr->state == DISPLAY_RUNNING)
	{
		Status = DisplayStop(dispPtr);
		if (Status != XST_SUCCESS)
		{
			xil_printf("Cannot change mode, unable to stop display %d\r\n", Status);
			return XST_FAILURE;
		}
	}

	dispPtr->vMode = *newMode;

	return XST_SUCCESS;
}
```

```
/* --------------------------------------------------------- */
/***		DisplayChangeFrame(DisplayCtrl *dispPtr, u32 frameIndex)
**		Parameters:
**			dispPtr - Pointer to the initialized DisplayCtrl struct
**			frameIndex - Index of the framebuffer to change to (must
**					be between 0 and (DISPLAY_NUM_FRAMES - 1))
**		Return Value: int
**			XST_SUCCESS if successful, XST_FAILURE otherwise
**		Errors:
**		Description:
**			Changes the frame currently being displayed.
*/

int DisplayChangeFrame(DisplayCtrl *dispPtr, u32 frameIndex)
{
   int Status;

   dispPtr->curFrame = frameIndex;
   /*
    * If currently running, then the DMA needs to be told to start reading from the desired frame
    * at the end of the current frame
    */
   if (dispPtr->state == DISPLAY_RUNNING)
   {
		Status = XAxiVdma_StartParking(&(dispPtr->vdma), dispPtr->curFrame, XAXIVDMA_READ);
		if (Status != XST_SUCCESS)
		{
			xil_printf("Cannot change frame, unable to start parking %d\r\n", Status);
			return XST_FAILURE;
		}
   }

   return XST_SUCCESS;
}

u32 DisplayClkCountCalc(u32 divide)
{
   u32 output = 0;
   u32 divCalc = 0;

   divCalc = DisplayClkDivider(divide);
   if (divCalc == ERR_CLKDIVIDER)
		output = ERR_CLKCOUNTCALC;
   else
		output = (0xFFF & divCalc) | ((divCalc << 10) & 0x00C00000);
   return output;
}

u32 DisplayClkDivider(u32 divide)
{
   u32 output = 0;
   u32 highTime = 0;
   u32 lowTime = 0;

   if ((divide < 1) || (divide > 128))
		return ERR_CLKDIVIDER;

   if (divide == 1)
		return 0x1041;

   highTime = divide / 2;
   if (divide & 0b1) //if divide is odd
   {
		lowTime = highTime + 1;
		output = 1 << CLK_BIT_WEDGE;
   }
```

```
    else
    {
            lowTime = highTime;
    }

    output |= 0x03F & lowTime;
    output |= 0xFC0 & (highTime << 6);
    return output;
}

u32 DisplayClkFindReg (ClkConfig *regValues, ClkMode *clkParams)
{
    if ((clkParams->fbmult < 2) || clkParams->fbmult > 64 )
            return 0;

    regValues->clk0L = DisplayClkCountCalc(clkParams->clkdiv);
    if (regValues->clk0L == ERR_CLKCOUNTCALC)
            return 0;

    regValues->clkFBL = DisplayClkCountCalc(clkParams->fbmult);
    if (regValues->clkFBL == ERR_CLKCOUNTCALC)
            return 0;

    regValues->clkFBH_clk0H = 0;

    regValues->divclk = DisplayClkDivider(clkParams->maindiv);
    if (regValues->divclk == ERR_CLKDIVIDER)
            return 0;

    regValues->lockL = (u32) (lock_lookup[clkParams->fbmult - 1] & 0xFFFFFFFF);

    regValues->fltr_lockH = (u32) ((lock_lookup[clkParams->fbmult - 1] >> 32) & 0x000000FF);
    regValues->fltr_lockH |= ((filter_lookup_low[clkParams->fbmult - 1] << 16) & 0x03FF0000);

    return 1;
}

void DisplayClkWriteReg (ClkConfig *regValues, u32 dispCtrlAddr)
{
    Xil_Out32(dispCtrlAddr + OFST_DISPLAY_CLK_L, regValues->clk0L);
    Xil_Out32(dispCtrlAddr + OFST_DISPLAY_FB_L, regValues->clkFBL);
    Xil_Out32(dispCtrlAddr + OFST_DISPLAY_FB_H_CLK_H, regValues->clkFBH_clk0H);
    Xil_Out32(dispCtrlAddr + OFST_DISPLAY_DIV, regValues->divclk);
    Xil_Out32(dispCtrlAddr + OFST_DISPLAY_LOCK_L, regValues->lockL);
    Xil_Out32(dispCtrlAddr + OFST_DISPLAY_FLTR_LOCK_H, regValues->fltr_lockH);
}

/*
 * TODO:This function currently requires that the reference clock is 100MHz.
 *          This should be changed so that the ref. clock can be specified, or read directly
 *          out of hardware.
 */
double DisplayClkFindParams(double freq, ClkMode *bestPick)
{
    double bestError = 2000.0;
    double curError;
    double curClkMult;
    double curFreq;
    u32 curDiv, curFb, curClkDiv;
    u32 minFb = 0;
    u32 maxFb = 0;

    bestPick->freq = 0.0;
/*
 * TODO: replace with a smarter algorithm that doesn't doesn't check every possible combination
 */
```

```
    for (curDiv = 1; curDiv <= 10; curDiv++)
    {
            minFb = curDiv * 6; //This accounts for the 100MHz input and the 600MHz minimum VCO
            maxFb = curDiv * 12; //This accounts for the 100MHz input and the 1200MHz maximum VCO
            if (maxFb > 64)
                    maxFb = 64;
//This multiplier is used to find the best clkDiv value for each FB value
            curClkMult = (100.0 / (double) curDiv) / freq;

            curFb = minFb;
            while (curFb <= maxFb)
            {
                    curClkDiv = (u32) ((curClkMult * (double)curFb) + 0.5);
                    curFreq = ((100.0 / (double) curDiv) / (double) curClkDiv) * (double) curFb;
                    curError = fabs(curFreq - freq);
                    if (curError < bestError)
                    {
                            bestError = curError;
                            bestPick->clkdiv = curClkDiv;
                            bestPick->fbmult = curFb;
                            bestPick->maindiv = curDiv;
                            bestPick->freq = curFreq;
                    }

                    curFb++;
            }
    }

    return bestError;
}
```

Display_ctrl.h

```
/************************************************************************/
	/*									*/
	/* display_ctrl.h	--	Digilent Display Controller Driver		*/
	/*									*/
	/************************************************************************/
	/* Author: Sam Bobrowicz						*/
	/* Copyright 2014, Digilent Inc.						*/
	/************************************************************************/
	/*  Module Description:							*/
	/*									*/
	/*		This module provides an easy to use API for controlling the	*/
	/*		Digilent display controller core (axi_dispctrl). It supports	*/
	/*		run-time resolution setting and seamless framebuffer-swapping	*/
	/*		for tear-free animation.					*/
	/*									*/
	/*		To use this driver, you must have an axi_dispctrl and axi_vdma	*/
	/*		core present in your system. For information on how to properly */
	/*		configure these cores within your design, refer to the		*/
	/*		axi_dispctrl data sheet accessible from Vivado and XPS.		*/
	/*									*/
	/*		The following steps should be followed to use this driver:	*/
	/*		1) Create a DisplayCtrl object and pass a pointer to it to	*/
	/*		   DisplayInitialize.						*/
	/*		2) Call DisplaySetMode to set the desired mode			*/
	/*		3) Call DisplayStart to begin outputting data to the display	*/
	/*		4) To create a seamless animation, draw the next image to a	*/
	/*		   framebuffer currently not being displayed. Then call		*/
	/*		   DisplayChangeFrame to begin displaying that frame.		*/
	/*		   Repeat as needed, only ever modifying inactive frames.	*/
	/*		5) To change the resolution, call DisplaySetMode, followed by	*/
	/*		   DisplayStart again.						*/
	/*									*/
	/*		This module contains code from the Xilinx Demo titled		*/
	/*		"xiicps_polled_master_example.c." Xilinx XAPP888 was also	*/
	/*		referenced for information on reconfiguring the MMCM or PLL.	*/
	/*		Note that functions beginning with "DisplayClk" are used	*/
	/*		internally for this purpose and should not need to be called	*/
	/*		externally.							*/
	/*									*/
	/************************************************************************/
	/*  Revision History:							*/
	/*									*/
	/*		2/20/2014(SamB): Created					*/
	/*									*/
	/************************************************************************/

	#ifndef DISPLAY_CTRL_H_
	#define DISPLAY_CTRL_H_
	/* ------------------------------------------------------------ */
	/*		Include File Definitions			*/
	/* ------------------------------------------------------------ */
	#include "xil_types.h"
	#include "vga_modes.h"
	#include "xaxivdma.h"
	/* ------------------------------------------------------------ */
	/*		Miscellaneous Declarations			*/
	/* ------------------------------------------------------------ */
	#define CLK_BIT_WEDGE 13
	#define CLK_BIT_NOCOUNT 12

	#define ERR_CLKCOUNTCALC 0xFFFFFFFF //This value is used to signal an error

	#define OFST_DISPLAY_CTRL 0x0
	#define OFST_DISPLAY_STATUS 0x4
```

```c
#define OFST_DISPLAY_VIDEO_START 0x8
#define OFST_DISPLAY_CLK_L 0x1C
#define OFST_DISPLAY_FB_L 0x20
#define OFST_DISPLAY_FB_H_CLK_H 0x24
#define OFST_DISPLAY_DIV 0x28
#define OFST_DISPLAY_LOCK_L 0x2C
#define OFST_DISPLAY_FLTR_LOCK_H 0x30

#define BIT_DISPLAY_RED 16
#define BIT_DISPLAY_BLUE 0
#define BIT_DISPLAY_GREEN 8

#define BIT_DISPLAY_START 0
#define BIT_DISPLAY_RUNNING 1

#define DISPLAY_NOT_HDMI 0
#define DISPLAY_HDMI 1

/* This driver currently supports 3 frames. */
#define DISPLAY_NUM_FRAMES 3

/* WEDGE and NOCOUNT can't both be high, so this is used to signal an error state */
#define ERR_CLKDIVIDER (1 << CLK_BIT_WEDGE | 1 << CLK_BIT_NOCOUNT)
/* ------------------------------------------------------------ */
/*                  General Type Declaration                    */
/* ------------------------------------------------------------ */
typedef enum {
   DISPLAY_STOPPED = 0,
   DISPLAY_RUNNING = 1
} DisplayState;

typedef struct {
		u32 clk0L;
		u32 clkFBL;
		u32 clkFBH_clk0H;
		u32 divclk;
		u32 lockL;
		u32 fltr_lockH;
} ClkConfig;

typedef struct {
		double freq;
		u32 fbmult;
		u32 clkdiv;
		u32 maindiv;
} ClkMode;

typedef struct {
   u32 dispCtrlAddr;          /*Physical Base address of the disp_ctrl core*/
   int fHdmi;           /*flag indicating if the display controller is being used to drive an HDMI transmitter*/
   XAxiVdma vdma;          /*VDMA driver struct*/
   XAxiVdma_DmaSetup vdmaConfig;       /*VDMA channel configuration*/
   VideoMode vMode;        /*Current Video mode*/
   u32 *framePtr[DISPLAY_NUM_FRAMES];       /* Array of pointers to the framebuffers */
   u32 stride;                /* The line stride of the framebuffers, in bytes */
   double pxlFreq;            /* Frequency of clock currently being generated */
   u32 curFrame;              /* Current frame being displayed */
   DisplayState state;        /* Indicates if the Display is currently running */
} DisplayCtrl;
/* ------------------------------------------------------------ */
/*                  Variable Declarations                       */
/* ------------------------------------------------------------ */
static const u64 lock_lookup[64] = {
  0b0011000110111110100011111010010000000001,
  0b0011000110111110100011111010010000000001,
  0b0100001000111110100011111010010000000001,
    0b0101101011111110100011111010010000000001,
    0b0111001110111110100011111010010000000001,
    0b1000110001111110100011111010010000000001,
    0b1001110011111110100011111010010000000001,
    0b1011010110111110100011111010010000000001,
    0b1100111001111110100011111010010000000001,
    0b1110011100111110100011111010010000000001,
    0b1111111111110000100111111010010000000001,
    0b1111111111100111001111111010010000000001,
    0b1111111111101110111011111010010000000001,
    0b1111111111101011110011111010010000000001,
    0b1111111111101000101011111010010000000001,
    0b1111111111100111000111111010010000000001,
    0b1111111111100011111111111010010000000001,
    0b1111111111100010011011111010010000000001,
    0b1111111111100000110111111010010000000001,
    0b1111111111011111010011111010010000000001,
    0b1111111111011101101111111010010000000001,
    0b1111111111011100001011111010010000000001,
    0b1111111111011010100111111010010000000001,
    0b1111111111011001000011111010010000000001,
    0b1111111111011001000011111010010000000001,
    0b1111111111010111011111111010010000000001,
    0b1111111111010101111011111010010000000001,
    0b1111111111010101111011111010010000000001,
    0b1111111111010100010111111010010000000001,
    0b1111111111010100010111111010010000000001,
    0b1111111111010010110011111010010000000001,
    0b1111111111010010110011111010010000000001,
    0b1111111111010010110011111010010000000001,
    0b1111111111010001001111111010010000000001,
    0b1111111111010001001111111010010000000001,
    0b1111111111010001001111111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001,
    0b1111111111001111101011111010010000000001
};

static const u32 filter_lookup_low[64] = {
    0b0001011111,
    0b0001010111,
        0b0001111011,
        0b0001011011,
        0b0001101011,
        0b0001110011,
        0b0001110011,
        0b0001110011,
        0b0001110011,
        0b0001001011,
        0b0001001011,
        0b0001001011,
        0b0010110011,
        0b0001010011,
        0b0001010011,
        0b0001010011,
        0b0001010011,
        0b0001010011,
        0b0001010011,
        0b0001010011,
        0b0001010011,
        0b0001010011,
        0b0001010011,
        0b0001100011,
        0b0001100011,
        0b0001100011,
        0b0001100011,
        0b0001100011,
        0b0001100011,
        0b0001100011,
        0b0001100011,
        0b0001100011,
        0b0001100011,
        0b0001100011,
        0b0001100011,
        0b0001100011,
        0b0001100011,
        0b0010010011,
        0b0010010011,
        0b0010010011,
        0b0010010011,
        0b0010010011,
        0b0010010011,
        0b0010010011,
        0b0010010011,
        0b0010010011,
        0b0010010011,
        0b0010100011,
        0b0010100011,
        0b0010100011,
        0b0010100011,
        0b0010100011,
        0b0010100011,
        0b0010100011,
        0b0010100011,
        0b0010100011,
        0b0010100011,
        0b0010100011,
        0b0010100011,
        0b0010100011,
        0b0010100011,
        0b0010100011,
        0b0010100011,
        0b0010100011
};
/* ----------------------------------------------------------- */
/*              Procedure Declarations                         */
/* ----------------------------------------------------------- */
```

```
u32 DisplayClkCountCalc(u32 divide);
u32 DisplayClkDivider(u32 divide);
u32 DisplayClkFindReg (ClkConfig *regValues, ClkMode *clkParams);
void DisplayClkWriteReg (ClkConfig *regValues, u32 dispCtrlAddr);
double DisplayClkFindParams(double freq, ClkMode *bestPick);

int DisplayStop(DisplayCtrl *dispPtr);
int DisplayStart(DisplayCtrl *dispPtr);
int DisplayInitialize(DisplayCtrl *dispPtr, u16 vdmaId, u32 dispCtrlAddr, int fHdmi, u32
*framePtr[DISPLAY_NUM_FRAMES], u32 stride);
int DisplaySetMode(DisplayCtrl *dispPtr, const VideoMode *newMode);
int DisplayChangeFrame(DisplayCtrl *dispPtr, u32 frameIndex);

/* ------------------------------------------------------------ */
/************************************************************************/
#endif /* DISPLAY_CTRL_H_ */
```

timer_ps.c

```
/******************************************************************************/
    /*                                                                    */
    /* timer_ps.c      --      Timer Delay    for Zynq systems            */
    /*                                                                    */
    /**********************************************************************/
    /* Author: Sam Bobrowicz                                              */
    /* Copyright 2014, Digilent Inc.                                      */
    /**********************************************************************/
    /*  Module Description:                                               */
    /*                                                                    */
    /*          Implements an accurate delay function using the scutimer. */
    /*          Code from this module will cause conflicts with other code that */
    /*          requires the Zynq's scu timer.                            */
    /*                                                                    */
    /*          This module contains code from the Xilinx Demo titled     */
    /*          "xscutimer_polled_example.c"                              */
    /*                                                                    */
    /**********************************************************************/
    /*  Revision History:                                                 */
    /*                                                                    */
    /*          2/14/2014(SamB): Created                                  */
    /*                                                                    */
    /**********************************************************************/


    /* ------------------------------------------------------------ */
    /*          Include File Definitions                            */
    /* ------------------------------------------------------------ */
    #include "timer_ps.h"
    #include "xscutimer.h"
    #include "xil_types.h"

    /* ------------------------------------------------------------ */
    /*          Global Variables                                    */
    /* ------------------------------------------------------------ */

    XScuTimer TimerInstance;/* Cortex A9 Scu Private Timer Instance */

    /* ------------------------------------------------------------ */
    /*          Procedure Definitions                               */
    /* ------------------------------------------------------------ */

    /***        TimerInitialize(u16 TimerDeviceId)
    **          Parameters:
    **                  TimerDeviceId - The DEVICE ID of the Zynq SCU TIMER
    **          Return Value: int
    **                  XST_SUCCESS if successful
    **          Errors:
    **          Description: Configures the global timer struct to access the
    **                   the SCU timer. Can be called multiple times without error.
    */
    int TimerInitialize(u16 TimerDeviceId)
    {
        int Status;
        XScuTimer *TimerInstancePtr = &TimerInstance;
        XScuTimer_Config *ConfigTmrPtr;

        /*
         * Initialize the Scu Private Timer driver.
         */
        ConfigTmrPtr = XScuTimer_LookupConfig(TimerDeviceId);

        /*
         * This is where the virtual address would be used, this example
```

```
     * uses physical address. Note that it is not considered an error
     * if the timer has already been initialized.
     */
    Status = XScuTimer_CfgInitialize(TimerInstancePtr, ConfigTmrPtr,
                        ConfigTmrPtr->BaseAddr);
    if (Status != XST_SUCCESS || Status != XST_DEVICE_IS_STARTED) {
            return XST_FAILURE;
    }

    /*
     * Set prescaler to 1
     */
    XScuTimer_SetPrescaler(TimerInstancePtr, 0);

    return Status;
}
/* ------------------------------------------------------------ */
/***        TimerDelay(u32 uSDelay)
**          Parameters:
**                  uSDelay - Desired delay in micro seconds
**          Return Value:
**          Errors:
**          Description: Blocks execution for the desired amount of time.
**                          TimerInitialize must have been called at least once before calling this function.
*/
/* ------------------------------------------------------------ */
void TimerDelay(u32 uSDelay)
{
    u32 timerCnt;

    timerCnt = (TIMER_FREQ_HZ / 1000000) * uSDelay;

    XScuTimer_Stop(&TimerInstance);
    XScuTimer_DisableAutoReload(&TimerInstance);
    XScuTimer_LoadTimer(&TimerInstance, timerCnt);
    XScuTimer_Start(&TimerInstance);
    while (XScuTimer_GetCounterValue(&TimerInstance))
    {}

    return;
}
```

timer_ps.h

```
/************************************************************************/
    /*                                                                  */
    /* timer_ps.h      --      Timer Delay      for Zynq systems        */
    /*                                                                  */
    /********************************************************************/
    /* Author: Sam Bobrowicz                                            */
    /* Copyright 2014, Digilent Inc.                                    */
    /********************************************************************/
    /*  Module Description:                                             */
    /*                                                                  */
    /*          Implements an accurate delay function using the scutimer.  */
    /*          Code from this module will cause conflicts with other code that */
    /*          requires the Zynq's scu timer.                          */
    /*                                                                  */
    /*          This module contains code from the Xilinx Demo titled   */
    /*          "xscutimer_polled_example.c"                            */
    /*                                                                  */
    /********************************************************************/
    /*  Revision History:                                               */
    /*                                                                  */
    /*          2/14/2014(SamB): Created                                */
    /*                                                                  */
    /********************************************************************/
    #ifndef TIMER_PS_H_
    #define TIMER_PS_H_

    #include "xil_types.h"
    #include "xparameters.h"

    /* ------------------------------------------------------------ */
    /*          Miscellaneous Declarations                          */
    /* ------------------------------------------------------------ */

    #define TIMER_FREQ_HZ (XPAR_CPU_CORTEXA9_0_CPU_CLK_FREQ_HZ / 2)

    /* ------------------------------------------------------------ */
    /*          Procedure Declarations                              */
    /* ------------------------------------------------------------ */

    int TimerInitialize(u16 TimerDeviceId);
    void TimerDelay(u32 uSDelay);

    /* ------------------------------------------------------------ */

    /********************************************************************/

    #endif /* TIMER_H_ */
```

vga_modes.h

```
/************************************************************************/
    /*                                                                  */
    /* vga_modes.h    --    VideoMode definitions                       */
    /*                                                                  */
    /************************************************************************/
    /* Author: Sam Bobrowicz                                            */
    /* Copyright 2014, Digilent Inc.                                    */
    /************************************************************************/
    /*  Module Description:                                             */
    /*                                                                  */
    /*          This file contains the definition of the VideoMode type, and */
    /*          also defines several common video modes                 */
    /*                                                                  */
    /************************************************************************/
    /*  Revision History:                                               */
    /*                                                                  */
    /*          2/17/2014(SamB): Created                                */
    /*                                                                  */
    /************************************************************************/

    #ifndef VGA_MODES_H_
    #define VGA_MODES_H_

    typedef struct {
       char label[64]; /* Label describing the resolution */
       u32 width; /*Width of the active video frame*/
       u32 height; /*Height of the active video frame*/
       u32 hps; /*Start time of Horizontal sync pulse, in pixel clocks (active width + H. front porch)*/
       u32 hpe; /*End time of Horizontal sync pulse, in pixel clocks (active width + H. front porch + H. sync width)*/
       u32 hmax; /*Total number of pixel clocks per line (active width + H. front porch + H. sync width + H. back porch) */
       u32 hpol; /*hsync pulse polarity*/
       u32 vps; /*Start time of Vertical sync pulse, in lines (active height + V. front porch)*/
       u32 vpe; /*End time of Vertical sync pulse, in lines (active height + V. front porch + V. sync width)*/
       u32 vmax; /*Total number of lines per frame (active height + V. front porch + V. sync width + V. back porch) */
       u32 vpol; /*vsync pulse polarity*/
       double freq; /*Pixel Clock frequency*/
    } VideoMode;

    static const VideoMode VMODE_640x480 = {
       .label = "640x480@60Hz",
       .width = 640,
       .height = 480,
       .hps = 656,
       .hpe = 752,
       .hmax = 799,
       .hpol = 0,
       .vps = 490,
       .vpe = 492,
       .vmax = 524,
       .vpol = 0,
       .freq = 25.0
    };

    static const VideoMode VMODE_800x600 = {
       .label = "800x600@60Hz",
       .width = 800,
       .height = 600,
       .hps = 840,
       .hpe = 968,
       .hmax = 1055,
       .hpol = 1,
       .vps = 601,
       .vpe = 605,
       .vmax = 627,
       .vpol = 1,
       .freq = 40.0
    };

    static const VideoMode VMODE_1280x1024 = {
       .label = "1280x1024@60Hz",
       .width = 1280,
       .height = 1024,
       .hps = 1328,
       .hpe = 1440,
       .hmax = 1687,
       .hpol = 1,
       .vps = 1025,
       .vpe = 1028,
       .vmax = 1065,
       .vpol = 1,
       .freq = 108.0
    };

    static const VideoMode VMODE_1280x720 = {
       .label = "1280x720@60Hz",
       .width = 1280,
       .height = 720,
       .hps = 1390,
       .hpe = 1430,
       .hmax = 1649,
       .hpol = 1,
       .vps = 725,
       .vpe = 730,
       .vmax = 749,
       .vpol = 1,
       .freq = 74.25, //74.2424 is close enough
    };

    static const VideoMode VMODE_1920x1080 = {
       .label = "1920x1080@60Hz",
       .width = 1920,
       .height = 1080,
       .hps = 2008,
       .hpe = 2052,
       .hmax = 2199,
       .hpol = 1,
       .vps = 1084,
       .vpe = 1089,
       .vmax = 1124,
       .vpol = 1,
       .freq = 148.5 //148.57 is close enough
    };

    #endif /* VGA_MODES_H_ */
```