












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
ccess	 audio_demo	4/8/2014 4:10 AM	C File	15 KB
op	 audio_demo	4/8/2014 4:10 AM	H File	3 KB
loads	 display_ctrl	4/8/2014 4:10 AM	C File	17 KB
nents	 display_ctrl	4/8/2014 4:10 AM	H File	11 KB
es	 display_demo	4/10/2014 1:34 PM	C File	14 KB
nents	 display_demo	4/8/2014 4:10 AM	H File	3 KB
	 lscript	4/8/2014 4:09 AM	LD File	7 KB
header	 main	4/11/2014 7:19 AM	C File	5 KB
e11_12 VGA ar	 timer_ps	4/8/2014 4:10 AM	C File	4 KB
	 timer_ps	4/8/2014 4:10 AM	H File	2 KB
61 ah10 -- VG	 vga_modes	4/8/2014 4:10 AM	H File	4 KB

main.c

```

1  /*****
2  /*
3  /* main.c  --   ZYBO Base System demonstration
4  /*
5  /*****
6  /* Author: Sam Bobrowicz
7  /* Copyright 2014, Digilent Inc.
8  /*****
9  /* Module Description:
10 /*
11 /*      This file contains code for running a demonstration of the
12 /*      Video output and audio capabilities of the ZYBO.
13 /*
14 /*****
15 /* Revision History:
16 /*
17 /*      2/25/2014(SamB): Created
18 /*
19 /*****
20
21 /* ----- */
22 /*      Include File Definitions
23 /* ----- */
24
25 #include "display_demo.h"
26 #include "audio_demo.h"
27 #include "timer_ps.h"
28 #include "xparameters.h"
29 #include "uartps.h"
30
31 /* ----- */
32 /*      Global Variables
33 /* ----- */
34
35 /*
36 /* Display Driver structs
37 /*
38 DisplayCtrl hdmiCtrl;
39 DisplayCtrl vgaCtrl;
40
41 /*
42 /* Framebuffers for each display device
43 /*
44 u32 vgaBuf[DISPLAY_NUM_FRAMES][DISPLAYDEMO_MAX_FRAME];
45 u32 hdmiBuf[DISPLAY_NUM_FRAMES][DISPLAYDEMO_MAX_FRAME];

```

Video Run 1920x1080 (21KTV)

u32 unsigned 32bits

page 21, line 84

page 11 1920x1080

```

46
47 /*
48  * XPAR redefines
49  */
50 #define VGA_BASEADDR XPAR_AXI_DISPCTRL_0_S_AXI_BASEADDR
51 #define HDMI_BASEADDR XPAR_AXI_DISPCTRL_1_S_AXI_BASEADDR
52 #define VGA_VDMA_ID XPAR_AXIVDMA_0_DEVICE_ID
53 #define HDMI_VDMA_ID XPAR_AXIVDMA_1_DEVICE_ID
54 #define AUDIO_IIC_ID XPAR_XIICPS_0_DEVICE_ID
55 #define AUDIO_CTRL_BASEADDR XPAR_AXI_I2S_ADI_1_S_AXI_BASEADDR
56 #define SCU_TIMER_ID XPAR_SCUTIMER_DEVICE_ID
57 #define UART_BASEADDR XPAR_PS7_UART_1_BASEADDR
58 #define SW_BASEADDR XPAR_SWS_4BITS_BASEADDR
59 #define BTN_BASEADDR XPAR_BTNS_4BITS_BASEADDR
60
61 void MainDemoPrintMenu();
62
63 /* ----- */
64 /*      Procedure Definitions      */
65 /* ----- */
66
67 int main(void)
68 {
69     u32 *vgaPtr[DISPLAY_NUM_FRAMES];
70     u32 *hdmiPtr[DISPLAY_NUM_FRAMES];
71     int i;
72     char userInput = 0;
73
74     for (i = 0; i < DISPLAY_NUM_FRAMES; i++)
75     {
76         vgaPtr[i] = vgaBuf[i];
77         hdmiPtr[i] = hdmiBuf[i];
78     }
79
80     DisplayDemoInitialize(&vgaCtrl, VGA_VDMA_ID, SCU_TIMER_ID, VGA_BASEADDR, DISPLAY_NOT_HDMI,
vgaPtr);
81     DisplayDemoInitialize(&hdmiCtrl, HDMI_VDMA_ID, SCU_TIMER_ID, HDMI_BASEADDR, DISPLAY_HDMI,
hdmiPtr);
82     AudioInitialize(SCU_TIMER_ID, AUDIO_IIC_ID, AUDIO_CTRL_BASEADDR);
83     TimerInitialize(SCU_TIMER_ID);
84
85     /* Flush UART FIFO */
86     while (XUartPs_IsReceiveData(UART_BASEADDR))
87     {
88         XUartPs_ReadReg(UART_BASEADDR, XUARTPS_FIFO_OFFSET);
89     }
90
91     while (userInput != 'q')
92     {
93         MainDemoPrintMenu();
94
95         /* Wait for data on UART */
96         while (!XUartPs_IsReceiveData(UART_BASEADDR))
97         {}
98
99         /* Store the first character in the UART recieve FIFO and echo it */
100        userInput = XUartPs_ReadReg(UART_BASEADDR, XUARTPS_FIFO_OFFSET);
101        xil_printf("%c", userInput);
102
103        switch (userInput)
104        {
105            case '1':
106                AudioRunDemo(AUDIO_CTRL_BASEADDR, UART_BASEADDR, SW_BASEADDR,
BTN_BASEADDR);
107                break;
108            case '2':

```

frame

} pointers

←

```

109         DisplayDemoRun(&vgaCtrl, UART_BASEADDR);
110         break;
111     case '3':
112         DisplayDemoRun(&hdmiCtrl, UART_BASEADDR);
113         break;
114     case 'q':
115         break;
116     default :
117         xil_printf("\n\rInvalid Selection");
118         TimerDelay(500000);
119     }
120 }
121
122 return 0;
123 }
124
125 void MainDemoPrintMenu()
126 {
127     xil_printf("\x1B[H"); //Set cursor to top left of terminal
128     xil_printf("\x1B[2J"); //Clear terminal
129     xil_printf("*****\n\r");
130     xil_printf("*****\n\r");
131     xil_printf("      ZYBO Base System User Demo      *\n\r");
132     xil_printf("*****\n\r");
133     xil_printf("*****\n\r");
134     xil_printf("\n\r");
135     xil_printf("1 - Audio Demo\n\r");
136     xil_printf("2 - VGA output demo\n\r");
137     xil_printf("3 - HDMI output demo\n\r");
138     xil_printf("q - Quit\n\r");
139     xil_printf("\n\r");
140     xil_printf("Select a demo to run:");
141 }
142 /* end main.c*/

```

display_demo.c

```
1  /*****
2  /*
3  /* display_demo.c --      ZYBO Display demonstration      */
4  /*
5  /*****
6  /* Author: Sam Bobrowicz      */
7  /* Copyright 2014, Digilent Inc.      */
8  /*****
9  /* Module Description:      */
10 /*
11 /*      This file contains code for running a demonstration of the      */
12 /*      Video output capabilities on the ZYBO. It is a good example of      */
13 /*      how to properly use the display_ctrl driver.      */
14 /*
15 /*      This module contains code from the Xilinx Demo titled      */
16 /*      "xiicps_polled_master_example.c"      */
17 /*
18 /*****
19 /* Revision History:      */
20 /*
21 /*      2/20/2014(SamB): Created      */
22 /*
23 /*****
24
25 /* ----- */
26 /*      Include File Definitions      */
27 /* ----- */
28
29 #include "display_demo.h"
30 #include <stdio.h>
31 #include "xuartps.h"
32 #include "math.h"
33 #include <ctype.h>
34 #include <stdlib.h>
35 #include "xil_types.h"
36 #include "xil_cache.h"
37 #include "timer_ps.h"
38 #include "display_ctrl.h"
39
40 /* ----- */
41 /*      Procedure Definitions      */
42 /* ----- */
43
44 int DisplayDemoInitialize(DisplayCtrl *dispPtr, u16 vdmaId, u16 timerId, u32 dispCtrlAddr, int fHdmi, u32
45 *framePtr[DISPLAY_NUM_FRAMES])
46 {
47     int Status;
48
49     TimerInitialize(timerId);
50
51     Status = DisplayInitialize(dispPtr, vdmaId, dispCtrlAddr, fHdmi, framePtr, DISPLAYDEMO_STRIDE);
52     if (Status != XST_SUCCESS)
53     {
54         xil_printf("Display Ctrl initialization failed during demo initialization%d\r\n", Status);
55         return XST_FAILURE;
56     }
57
58     Status = DisplayStart(dispPtr);
59     if (Status != XST_SUCCESS)
60     {
61         xil_printf("Couldn't start display during demo initialization%d\r\n", Status);
62         return XST_FAILURE;
63     }
64 }
```

```

64     DisplayDemoPrintTest(dispptr->framePtr[dispptr->curFrame], dispptr->vMode.width, dispptr->vMode.height, dispptr-
    >stride, DISPLAYDEMO_PATTERN_1);
65
66     return XST_SUCCESS;
67 }
68
69 void DisplayDemoPrintMenu(DisplayCtrl *dispptr)
70 {
71     xil_printf("\x1B[H"); //Set cursor to top left of terminal
72     xil_printf("\x1B[2J"); //Clear terminal
73     xil_printf("*****\n\r");
74     xil_printf("    ZYBO Display User Demo    *\n\r");
75     xil_printf("*****\n\r");
76     xil_printf("Port: %42s*\n\r", (dispptr->fHdmi == DISPLAY_HDMI) ? "HDMI" : "VGA");
77     xil_printf("Current Resolution: %28s*\n\r", dispptr->vMode.label);
78     printf("Pixel Clock Freq. (MHz): %23.3f*\n\r", (dispptr->fHdmi == DISPLAY_HDMI) ? (dispptr->pxlFreq / 5.0) :
    (dispptr->pxlFreq));
79     xil_printf("Current Frame Index: %27d*\n\r", dispptr->curFrame);
80     xil_printf("*****\n\r");
81     xil_printf("\n\r");
82     xil_printf("1 - Change Resolution\n\r");
83     xil_printf("2 - Change Frame\n\r");
84     xil_printf("3 - Print Blended Test Pattern to current Frame\n\r");
85     xil_printf("4 - Print Color Bar Test Pattern to current Frame\n\r");
86     xil_printf("5 - Invert Current Frame colors\n\r");
87     xil_printf("6 - Invert Current Frame colors seamlessly*\n\r");
88     xil_printf("q - Quit\n\r");
89     xil_printf("\n\r");
90     xil_printf("Note that option 6 causes the current frame index to be\n\r");
91     xil_printf("incremented. This is because the inverted frame is drawn\n\r");
92     xil_printf("to an inactive frame. After the drawing is complete, this\n\r");
93     xil_printf("frame is then set to be the active frame. This demonstrates\n\r");
94     xil_printf("how to properly update what is being displayed without image\n\r");
95     xil_printf("tearing. Options 3-5 all draw to the currently active frame,\n\r");
96     xil_printf("which is why not all pixels appear to be updated at once.\n\r");
97     xil_printf("\n\r");
98     xil_printf("Enter a selection:");
99 }
100
101 int DisplayDemoRun(DisplayCtrl *dispptr, u32 uartAddr)
102 {
103     char userInput = 0;
104     int nextFrame = 0;
105
106     /* Flush UART FIFO */
107     while (XUartPs_IsReceiveData(uartAddr))
108     {
109         XUartPs_ReadReg(uartAddr, XUARTPS_FIFO_OFFSET);
110     }
111
112     while (userInput != 'q')
113     {
114         DisplayDemoPrintMenu(dispptr);
115
116         /* Wait for data on UART */
117         while (!XUartPs_IsReceiveData(uartAddr))
118             {}
119
120         /* Store the first character in the UART receive FIFO and echo it */
121         userInput = XUartPs_ReadReg(uartAddr, XUARTPS_FIFO_OFFSET);
122         xil_printf("%c", userInput);
123
124         switch (userInput)
125         {
126             case '1':
127                 DisplayDemoChangeRes(dispptr, uartAddr);

```

```

128         break;
129     case '2':
130         nextFrame = dispPtr->curFrame + 1;
131         if (nextFrame >= DISPLAY_NUM_FRAMES)
132         {
133             nextFrame = 0;
134         }
135         DisplayChangeFrame(dispPtr, nextFrame);
136         break;
137     case '3':
138         DisplayDemoPrintTest(dispPtr->framePtr[dispPtr->curFrame], dispPtr->vMode.width, dispPtr-
>vMode.height, dispPtr->stride, DISPLAYDEMO_PATTERN_0);
139         break;
140     case '4':
141         DisplayDemoPrintTest(dispPtr->framePtr[dispPtr->curFrame], dispPtr->vMode.width, dispPtr-
>vMode.height, dispPtr->stride, DISPLAYDEMO_PATTERN_1);
142         break;
143     case '5':
144         DisplayDemoInvertFrame(dispPtr->framePtr[dispPtr->curFrame], dispPtr->framePtr[dispPtr->curFrame],
dispPtr->vMode.width, dispPtr->vMode.height, dispPtr->stride);
145         break;
146     case '6':
147         nextFrame = dispPtr->curFrame + 1;
148         if (nextFrame >= DISPLAY_NUM_FRAMES)
149         {
150             nextFrame = 0;
151         }
152         DisplayDemoInvertFrame(dispPtr->framePtr[dispPtr->curFrame], dispPtr->framePtr[nextFrame],
dispPtr->vMode.width, dispPtr->vMode.height, dispPtr->stride);
153         DisplayChangeFrame(dispPtr, nextFrame);
154         break;
155     case 'q':
156         break;
157     default :
158         xil_printf("\n\rInvalid Selection");
159         TimerDelay(500000);
160     }
161 }
162
163 return XST_SUCCESS;
164 }
165
166 void DisplayDemoChangeRes(DisplayCtrl *dispPtr, u32 uartAddr)
167 {
168     char userInput = 0;
169     int fResSet = 0;
170     int status;
171
172     /* Flush UART FIFO */
173     while (XUartPs_IsReceiveData(uartAddr))
174     {
175         XUartPs_ReadReg(uartAddr, XUARTPS_FIFO_OFFSET);
176     }
177
178     while (!fResSet)
179     {
180         DisplayDemoCRMMenu(dispPtr);
181
182         /* Wait for data on UART */
183         while (!XUartPs_IsReceiveData(uartAddr))
184         {}
185
186         /* Store the first character in the UART recieve FIFO and echo it */
187         userInput = XUartPs_ReadReg(uartAddr, XUARTPS_FIFO_OFFSET);
188         xil_printf("%c", userInput);
189         status = XST_SUCCESS;

```

```

190     switch (userInput)
191     {
192     case '1':
193         status = DisplayStop(disPptr);
194         DisplaySetMode(disPptr, &VMODE_640x480);
195         DisplayStart(disPptr);
196         fResSet = 1;
197         break;
198     case '2':
199         status = DisplayStop(disPptr);
200         DisplaySetMode(disPptr, &VMODE_800x600);
201         DisplayStart(disPptr);
202         fResSet = 1;
203         break;
204     case '3':
205         status = DisplayStop(disPptr);
206         DisplaySetMode(disPptr, &VMODE_1280x720);
207         DisplayStart(disPptr);
208         fResSet = 1;
209         break;
210     case '4':
211         status = DisplayStop(disPptr);
212         DisplaySetMode(disPptr, &VMODE_1280x1024);
213         DisplayStart(disPptr);
214         fResSet = 1;
215         break;
216     case '5':
217         status = DisplayStop(disPptr);
218         DisplaySetMode(disPptr, &VMODE_1920x1080);
219         DisplayStart(disPptr);
220         fResSet = 1;
221         break;
222     case 'q':
223         fResSet = 1;
224         break;
225     default :
226         xil_printf("\n\rInvalid Selection");
227         TimerDelay(500000);
228     }
229     if (status == XST_DMA_ERROR)
230     {
231         xil_printf("\n\rWARNING: AXI VDMA Error detected and cleared\n\r");
232     }
233 }
234 }
235
236 void DisplayDemoCRMMenu(DisplayCtrl *disPptr)
237 {
238     xil_printf("\x1B[H"); //Set cursor to top left of terminal
239     xil_printf("\x1B[2J"); //Clear terminal
240     xil_printf("*****\n\r");
241     xil_printf("    ZYBO Display User Demo    *\n\r");
242     xil_printf("*****\n\r");
243     xil_printf("Port: %42s*\n\r", (disPptr->fHdmi == DISPLAY_HDMI) ? "HDMI" : "VGA");
244     xil_printf("Current Resolution: %28s*\n\r", disPptr->vMode.label);
245     printf("Pixel Clock Freq. (MHz): %23.3f*\n\r", (disPptr->fHdmi == DISPLAY_HDMI) ? (disPptr->pxlFreq / 5.0) :
(disPptr->pxlFreq));
246     xil_printf("*****\n\r");
247     xil_printf("\n\r");
248     xil_printf("1 - %s\n\r", VMODE_640x480.label);
249     xil_printf("2 - %s\n\r", VMODE_800x600.label);
250     xil_printf("3 - %s\n\r", VMODE_1280x720.label);
251     xil_printf("4 - %s\n\r", VMODE_1280x1024.label);
252     xil_printf("5 - %s\n\r", VMODE_1920x1080.label);
253     xil_printf("q - Quit (don't change resolution)\n\r");
254     xil_printf("\n\r");

```

```

255     xil_printf("Select a new resolution:");
256 }
257
258 void DisplayDemoInvertFrame(u32 *srcFrame, u32 *destFrame, u32 width, u32 height, u32 stride)
259 {
260     u32 xcoi, ycoi;
261     u32 lineStart = 0;
262     for(ycoi = 0; ycoi < height; ycoi++)
263     {
264         for(xcoi = 0; xcoi < width; xcoi++)
265         {
266             destFrame[xcoi + lineStart] = ~srcFrame[xcoi + lineStart];
267         }
268         lineStart += stride / 4; /*The stride is in bytes, so it needs to be divided by four to get the u32 address*/
269     }
270     /* Flush the framebuffer memory range to ensure changes are written to the
271     * actual memory, and therefore accessible by the VDMA. */
272     Xil_DCacheFlushRange((unsigned int) destFrame, DISPLAYDEMO_MAX_FRAME * 4);
273 }
274
275
276 void DisplayDemoPrintTest(u32 *frame, u32 width, u32 height, u32 stride, int pattern)
277 {
278     u32 xcoi, ycoi;
279     u32 iPixelAddr;
280     u32 wStride;
281     u32 wRed, wBlue, wGreen, wColor;
282     u32 wCurrentInt;
283     double fRed, fBlue, fGreen, fColor;
284     u32 xLeft, xMid, xRight, xInt;
285     u32 yMid, yInt;
286     double xInc, yInc;
287
288     switch (pattern)
289     {
290     case DISPLAYDEMO_PATTERN_0:
291
292         wStride = stride / 4; /* Find the stride in 32-bit words */
293
294         xInt = width / 4; //Four intervals, each with width/4 pixels
295         xLeft = xInt;
296         xMid = xInt * 2;
297         xRight = xInt * 3;
298         xInc = 256.0 / ((double) xInt); //256 color intensities are cycled through per interval (overflow must be caught
when color=256.0)
299
300         yInt = height / 2; //Two intervals, each with width/2 lines
301         yMid = yInt;
302         yInc = 256.0 / ((double) yInt); //256 color intensities are cycled through per interval (overflow must be caught
when color=256.0)
303
304         fBlue = 0.0;
305         fRed = 256.0;
306         for(xcoi = 0; xcoi < width; xcoi++)
307         {
308             /*
309             * Convert color intensities to integers < 256, and trim values >=256
310             */
311             wRed = (fRed >= 256.0) ? 255 : ((u32) fRed);
312             wBlue = (fBlue >= 256.0) ? 255 : ((u32) fBlue);
313
314             wColor = (wRed << BIT_DISPLAY_RED) | (wBlue << BIT_DISPLAY_BLUE);
315             iPixelAddr = xcoi;
316             fGreen = 0.0;
317             for(ycoi = 0; ycoi < height; ycoi++)
318             {

```



```

319         wGreen = (fGreen >= 256.0) ? 255 : ((u32) fGreen);
320         frame[iPixelAddr] = wColor | (wGreen << BIT_DISPLAY_GREEN);
321         if (ycoi < yMid)
322         {
323             fGreen += yInc;
324         }
325         else
326         {
327             fGreen -= yInc;
328         }
329
330     /* This pattern is printed one vertical line at a time, so the address must be incremented
331     * by the stride instead of just 1. */
332     iPixelAddr += wStride;
333 }
334
335     if (xcoi < xLeft)
336     {
337         fBlue = 0.0;
338         fRed = xInc;
339     }
340     else if (xcoi < xMid)
341     {
342         fBlue += xInc;
343         fRed += xInc;
344     }
345     else if (xcoi < xRight)
346     {
347         fBlue -= xInc;
348         fRed = xInc;
349     }
350     else
351     {
352         fBlue += xInc;
353         fRed = 0;
354     }
355 }
356 /*Flush the framebuffer memory range to ensure changes are written to the
357 * actual memory, and therefore accessible by the VDMA. */
358 Xil_DCacheFlushRange((unsigned int) frame, DISPLAYDEMO_MAX_FRAME * 4);
359 break;
360 case DISPLAYDEMO_PATTERN_1:
361     wStride = stride / 4; /* Find the stride in 32-bit words */
362
363     xInt = width / 7; //Seven intervals, each with width/7 pixels
364     xInc = 256.0 / ((double) xInt); //256 color intensities per interval. Notice that overflow is handled for this pattern.
365
366     fColor = 0.0;
367     wCurrentInt = 1;
368     for(xcoi = 0; xcoi < width; xcoi++)
369     {
370         if (wCurrentInt & 0b001)
371             fRed = fColor;
372         else
373             fRed = 0.0;
374
375         if (wCurrentInt & 0b010)
376             fBlue = fColor;
377         else
378             fBlue = 0.0;
379
380         if (wCurrentInt & 0b100)
381             fGreen = fColor;
382         else
383             fGreen = 0.0;
384

```

```

385         /* Just draw white in the last partial interval (when width is not divisible by 7) */
386         if (wCurrentInt > 7)
387         {
388             wColor = 0x00FFFFFF;
389         }
390         else
391         {
392             wColor = ((u32) fRed << BIT_DISPLAY_RED) | ((u32) fBlue << BIT_DISPLAY_BLUE) | (
(u32) fGreen << BIT_DISPLAY_GREEN);
393         }
394         iPixelAddr = xcoi;
395
396         for(ycoi = 0; ycoi < height; ycoi++)
397         {
398             frame[iPixelAddr] = wColor;
399             /* This pattern is printed one vertical line at a time, so the address must be incremented
400             by the stride instead of just 1. */
401             iPixelAddr += wStride;
402         }
403
404         fColor += xInc;
405         if (fColor >= 256.0)
406         {
407             fColor = 0.0;
408             wCurrentInt++;
409         }
410     }
411     /* Flush the framebuffer memory range to ensure changes are written to the
412     * actual memory, and therefore accessible by the VDMA. */
413     Xil_DCacheFlushRange((unsigned int) frame, DISPLAYDEMO_MAX_FRAME * 4);
414     break;
415     default :
416         xil_printf("Error: invalid pattern passed to DisplayDemoPrintTest");
417     }
418 }
419 /*end display_demo.c */

```

display_demo.h

```
1  /*****
2  /*
3  /* display_demo.h --      ZYBO Display demonstration      */
4  /*
5  /*****
6  /* Author: Sam Bobrowicz      */
7  /* Copyright 2014, Digilent Inc.      */
8  /*****
9  /* Module Description:      */
10 /*
11 /*      This file contains code for running a demonstration of the      */
12 /*      Video output capabilities on the ZYBO. It is a good example of      */
13 /*      how to properly use the display_ctrl driver.      */
14 /*
15 /*      This module contains code from the Xilinx Demo titled      */
16 /*      "xiicps_polled_master_example.c"      */
17 /*
18 /*****
19 /* Revision History:      */
20 /*
21 /*      2/20/2014(SamB): Created      */
22 /*
23 /*****
24
25 #ifndef DISPLAY_DEMO_H_
26 #define DISPLAY_DEMO_H_
27
28 /* ----- */
29 /*      Include File Definitions      */
30 /* ----- */
31
32 #include "xil_types.h"
33 #include "display_ctrl.h"
34
35 /* ----- */
36 /*      Miscellaneous Declarations      */
37 /* ----- */
38
39 #define DISPLAYDEMO_PATTERN_0 0
40 #define DISPLAYDEMO_PATTERN_1 1
41
42 #define DISPLAYDEMO_MAX_FRAME (1920*1080)
43 #define DISPLAYDEMO_STRIDE (1920 * 4)
44
45 /* ----- */
46 /*      Procedure Declarations      */
47 /* ----- */
48
49 int DisplayDemoInitialize(DisplayCtrl *dispPtr, u16 vdmaId, u16 timerId, u32 dispCtrlAddr, int fHdmi, u32
    *framePtr[DISPLAY_NUM_FRAMES]);
50 void DisplayDemoPrintMenu(DisplayCtrl *dispPtr);
51 int DisplayDemoRun(DisplayCtrl *dispPtr, u32 uartAddr);
52 void DisplayDemoChangeRes(DisplayCtrl *dispPtr, u32 uartAddr);
53 void DisplayDemoCRMenu(DisplayCtrl *dispPtr);
54 void DisplayDemoInvertFrame(u32 *srcFrame, u32 *destFrame, u32 width, u32 height, u32 stride);
55 void DisplayDemoPrintTest(u32 *frame, u32 width, u32 height, u32 stride, int pattern);
56
57 /* ----- */
58 /*****
59 #endif /* DISPLAY_DEMO_H_ */
```

display_ctrl.c

```
1  /*****
2  /*
3  /* display_ctrl.c    --    Digilent Display Controller Driver
4  /*
5  /*****
6  /* Author: Sam Bobrowicz
7  /* Copyright 2014, Digilent Inc.
8  /*****
9  /* Module Description:
10 /*
11 /*      This module provides an easy to use API for controlling the
12 /*      Digilent display controller core (axi_dispctrl). It supports
13 /*      run-time resolution setting and seamless framebuffer-swapping
14 /*      for tear-free animation.
15 /*
16 /*      To use this driver, you must have an axi_dispctrl and axi_vdma
17 /*      core present in your system. For information on how to properly */
18 /*      configure these cores within your design, refer to the
19 /*      axi_dispctrl data sheet accessible from Vivado and XPS.
20 /*
21 /*      The following steps should be followed to use this driver:
22 /*      1) Create a DisplayCtrl object and pass a pointer to it to
23 /*      DisplayInitialize.
24 /*      2) Call DisplaySetMode to set the desired mode
25 /*      3) Call DisplayStart to begin outputting data to the display
26 /*      4) To create a seamless animation, draw the next image to a
27 /*      framebuffer currently not being displayed. Then call
28 /*      DisplayChangeFrame to begin displaying that frame.
29 /*      Repeat as needed, only ever modifying inactive frames.
30 /*      5) To change the resolution, call DisplaySetMode, followed by
31 /*      DisplayStart again.
32 /*
33 /*      This module contains code from the Xilinx Demo titled
34 /*      "xiicps_polled_master_example.c." Xilinx XAPP888 was also
35 /*      referenced for information on reconfiguring the MMCM or PLL.
36 /*      Note that functions beginning with "DisplayCk" are used
37 /*      internally for this purpose and should not need to be called
38 /*      externally.
39 /*****
40 /* Revision History:
41 /*
42 /*      2/20/2014(SamB): Created
43 /*
44 /*****
45 /*
46 /*      * TODO: Functionality needs to be added to take advantage of the MMCM's fractional
47 /*      * divide. This will allow a far greater number of frequencies to be synthesized. */
48 /*      ----- */
49 /*      Include File Definitions */
50 /*      ----- */
51 #include "display_ctrl.h"
52 #include "xil_io.h"
53 #include "math.h"
54 #include "xil_types.h"
55 #include "vga_modes.h"
56 #include "xaxivdma.h"
57 #include <stdio.h>
58 /*      ----- */
59 /*      Procedure Definitions */
60 /*      ----- */
61 /***      DisplayStop(DisplayCtrl *dispPtr)
62 **
63 **      Parameters:
64 **      dispPtr - Pointer to the initialized DisplayCtrl struct
```

```

65  **
66  **      Return Value: int
67  **          XST_SUCCESS if successful.
68  **          XST_DMA_ERROR if an error was detected on the DMA channel. The
69  **          Display is still successfully stopped, and the error is
70  **          cleared so that subsequent DisplayStart calls will be
71  **          successful. This typically indicates insufficient bandwidth
72  **          on the AXI Memory-Map Interconnect (VDMA<->DDR)
73  **
74  **      Description:
75  **          Halts output to the display
76  */
77  int DisplayStop(DisplayCtrl *dispPtr)
78  {
79      /*      * If already stopped, do nothing      */
80      if (dispPtr->state == DISPLAY_STOPPED)
81      {
82          return XST_SUCCESS;
83      }
84
85      /*
86       * Disable the disp_ctrl core, and wait for the current frame to finish (the core cannot stop mid-frame)
87       */
88      Xil_Out32(dispPtr->dispCtrlAddr + OFST_DISPLAY_CTRL, 0);
89      while (Xil_In32(dispPtr->dispCtrlAddr + OFST_DISPLAY_STATUS) & (1 << BIT_DISPLAY_RUNNING));
90
91      /*      * Stop the VDMA core      */
92      XAxiVdma_DmaStop(&(dispPtr->vdma), XAXIVDMA_READ);
93      while(XAxiVdma_IsBusy(&(dispPtr->vdma), XAXIVDMA_READ));
94
95      /*      * Update Struct state      */
96      dispPtr->pxlFreq = 0;
97      dispPtr->state = DISPLAY_STOPPED;
98
99      if (XAxiVdma_GetDmaChannelErrors(&(dispPtr->vdma), XAXIVDMA_READ))
100      {
101          XAxiVdma_ClearDmaChannelErrors(&(dispPtr->vdma), XAXIVDMA_READ, 0xFFFFFFFF);
102          return XST_DMA_ERROR;
103      }
104
105      return XST_SUCCESS;
106  }
107  /* ----- */
108  /***      DisplayStart(DisplayCtrl *dispPtr)
109  **
110  **      Parameters:
111  **          dispPtr - Pointer to the initialized DisplayCtrl struct
112  **      Return Value: int
113  **          XST_SUCCESS if successful, XST_FAILURE otherwise
114  **      Errors:
115  **      Description:
116  **          Starts the display.
117  */
118  int DisplayStart(DisplayCtrl *dispPtr)
119  {
120      int Status;
121      u32 vgaReg[5];
122      ClkConfig clkReg;
123      ClkMode clkMode;
124      int i;
125      double pxlClkFreq;
126
127      /*
128       * If already started, do nothing
129       */
130      if (dispPtr->state == DISPLAY_RUNNING)

```

```

131     {
132         return XST_SUCCESS;
133     }
134
135     /* Configure the VDMA to access a frame with the same dimensions as the current mode */
136     dispPtr->vdmaConfig.VertSizeInput = dispPtr->vMode.height;
137     dispPtr->vdmaConfig.HoriSizeInput = (dispPtr->vMode.width) * 4;
138     dispPtr->vdmaConfig.FixedFrameStoreAddr = dispPtr->curFrame;
139     /*
140      * Also reset the stride and address values, in case the user manually changed them
141      */
142     dispPtr->vdmaConfig.Stride = dispPtr->stride;
143     for (i = 0; i < DISPLAY_NUM_FRAMES; i++)
144     {
145         dispPtr->vdmaConfig.FrameStoreStartAddr[i] = (u32) dispPtr->framePtr[i];
146     }
147
148     /*
149      * Perform the VDMA driver calls required to start a transfer. Note that no data is actually
150      * transferred until the disp_ctrl core signals the VDMA core by pulsing fsync.
151      */
152     Status = XAxiVdma_DmaConfig(&(dispPtr->vdma), XAXIVDMA_READ, &(dispPtr->vdmaConfig));
153     if (Status != XST_SUCCESS)
154     {
155         xil_printf("Read channel config failed %d\r\n", Status);
156         return XST_FAILURE;
157     }
158     Status = XAxiVdma_DmaSetBufferAddr(&(dispPtr->vdma), XAXIVDMA_READ, dispPtr-
159     >vdmaConfig.FrameStoreStartAddr);
160     if (Status != XST_SUCCESS)
161     {
162         xil_printf("Read channel set buffer address failed %d\r\n", Status);
163         return XST_FAILURE;
164     }
165     Status = XAxiVdma_DmaStart(&(dispPtr->vdma), XAXIVDMA_READ);
166     if (Status != XST_SUCCESS)
167     {
168         xil_printf("Start read transfer failed %d\r\n", Status);
169         return XST_FAILURE;
170     }
171     Status = XAxiVdma_StartParking(&(dispPtr->vdma), dispPtr->curFrame, XAXIVDMA_READ);
172     if (Status != XST_SUCCESS)
173     {
174         xil_printf("Unable to park the channel %d\r\n", Status);
175         return XST_FAILURE;
176     }
177
178     /* Configure the disp_ctrl core with the display mode timing parameters */
179     vgaReg[0] = (dispPtr->vMode.width << 16) | (dispPtr->vMode.height);
180     vgaReg[1] = (dispPtr->vMode.hps << 16) | (dispPtr->vMode.hpe);
181     vgaReg[2] = (dispPtr->vMode.hpol << 16) | (dispPtr->vMode.hmax);
182     vgaReg[3] = (dispPtr->vMode.vps << 16) | (dispPtr->vMode.vpe);
183     vgaReg[4] = (dispPtr->vMode.vpol << 16) | (dispPtr->vMode.vmax);
184     for (i = 0; i < 5; i++)
185     {
186         Xil_Out32(dispPtr->dispCtrlAddr + OFST_DISPLAY_VIDEO_START + (i * 4), vgaReg[i]);
187     }
188
189     /* Calculate the PLL divider parameters based on the required pixel clock frequency */
190     if (dispPtr->fHdmi == DISPLAY_HDMI)
191     {
192         pxlClkFreq = dispPtr->vMode.freq * 5;
193     }
194     else
195     {
196         pxlClkFreq = dispPtr->vMode.freq;
197     }

```

```

196     }
197     DisplayClkFindParams(pxIFreq, &clkMode);
198
199     /*
200     * Store the obtained frequency to pxIFreq. It is possible that the PLL was not able to
201     * exactly generate the desired pixel clock, so this may differ from vMode.freq. */
202     dispPtr->pxIFreq = clkMode.freq;
203
204     /*
205     * Write to the PLL dynamic configuration registers to configure it with the calculated parameters.
206     */
207     if (!DisplayClkFindReg(&clkReg, &clkMode))
208     {
209         xil_printf("Error calculating CLK register values\n\r");
210         return XST_FAILURE;
211     }
212     DisplayClkWriteReg(&clkReg, dispPtr->dispCtrlAddr);
213
214     /*
215     * Enable the disp_ctrl core, which will signal the VDMA to begin transferring data
216     */
217     Xil_Out32(dispPtr->dispCtrlAddr + OFST_DISPLAY_CTRL, (1 << BIT_DISPLAY_START));
218
219     dispPtr->state = DISPLAY_RUNNING;
220
221     return XST_SUCCESS;
222 }
223
224 /* ----- */
225 /**
226  * DisplayInitialize(DisplayCtrl *dispPtr, u16 vdmaId, u32 dispCtrlAddr, int fHdmi, u32
227  * framePtr[DISPLAY_NUM_FRAMES], u32 stride)
228  **
229  * Parameters:
230  **
231  *     dispPtr - Pointer to the struct that will be initialized
232  *     vdmaId - DEVICE ID of the attached VDMA core
233  *     dispCtrlAddr - BASE ADDRESS of the axi_dispctrl core
234  *     fHdmi - flag indicating if the C_USE_BUFR_DIV5 parameter is set for the axi_dispctrl core.
235  *     Use DISPLAY_HDMI if it is set, otherwise use DISPLAY_NOT_HDMI
236  *     framePtr - array of pointers to the framebuffers. The framebuffers must be instantiated above this driver
237  *     stride - line stride of the framebuffers. This is the number of bytes between the start of one line and the start of another.
238  **
239  * Return Value: int
240  **
241  *     XST_SUCCESS if successful, XST_FAILURE otherwise
242  **
243  * Errors:
244  **
245  * Description:
246  **
247  *     Initializes the driver struct for use.
248  */
249 int DisplayInitialize(DisplayCtrl *dispPtr, u16 vdmaId, u32 dispCtrlAddr, int fHdmi, u32
250 framePtr[DISPLAY_NUM_FRAMES], u32 stride)
251 {
252     int Status;
253     int i;
254     XAxiVdma_Config *Config;
255
256     /*
257     * Initialize all the fields in the DisplayCtrl struct
258     */
259     dispPtr->curFrame = 0;
260     dispPtr->dispCtrlAddr = dispCtrlAddr;
261     dispPtr->fHdmi = fHdmi;
262     for (i = 0; i < DISPLAY_NUM_FRAMES; i++)
263     {
264         dispPtr->framePtr[i] = framePtr[i];
265     }
266     dispPtr->pxIFreq = 0;
267     dispPtr->state = DISPLAY_STOPPED;
268     dispPtr->stride = stride;

```

```

260     dispPtr->vMode = VMODE_640x480;
261
262     /*      * Initialize VDMA driver      */
263     Config = XAxiVdma_LookupConfig(vdmaId);
264     if (!Config)
265     {
266         xil_printf("No video DMA found for ID %d\r\n", vdmaId);
267         return XST_FAILURE;
268     }
269     Status = XAxiVdma_CfgInitialize(&(dispPtr->vdma), Config, Config->BaseAddress);
270     if (Status != XST_SUCCESS)
271     {
272         xil_printf("Configuration Initialization failed %d\r\n", Status);
273         return XST_FAILURE;
274     }
275
276     Status = XAxiVdma_SetFrmStore(&(dispPtr->vdma), DISPLAY_NUM_FRAMES, XAXIVDMA_READ);
277     if (Status != XST_SUCCESS)
278     {
279         xil_printf("Setting Frame Store Number Failed in Read Channel %d\r\n", Status);
280         return XST_FAILURE;
281     }
282
283     /*      * Initialize the VDMA Read configuration struct      */
284     dispPtr->vdmaConfig.FrameDelay = 0;
285     dispPtr->vdmaConfig.EnableCircularBuf = 1;
286     dispPtr->vdmaConfig.EnableSync = 0;
287     dispPtr->vdmaConfig.PointNum = 0;
288     dispPtr->vdmaConfig.EnableFrameCounter = 0;
289
290     return XST_SUCCESS;
291 }
292 /*-----*/
293 /**      DisplaySetMode(DisplayCtrl *dispPtr, const VideoMode *newMode)
294 **      Parameters:
295 **          dispPtr - Pointer to the initialized DisplayCtrl struct
296 **          newMode - The VideoMode struct describing the new mode.
297 **      Return Value: int
298 **          XST_SUCCESS if successful, XST_FAILURE otherwise
299 **      Errors:
300 **      Description:
301 **          Changes the resolution being output to the display. If the display
302 **          is currently started, it is automatically stopped (DisplayStart must
303 **          be called again).
304 */
305 int DisplaySetMode(DisplayCtrl *dispPtr, const VideoMode *newMode)
306 {
307     int Status;
308
309     /*
310      * If currently running, stop
311      */
312     if (dispPtr->state == DISPLAY_RUNNING)
313     {
314         Status = DisplayStop(dispPtr);
315         if (Status != XST_SUCCESS)
316         {
317             xil_printf("Cannot change mode, unable to stop display %d\r\n", Status);
318             return XST_FAILURE;
319         }
320     }
321
322     dispPtr->vMode = *newMode;
323
324     return XST_SUCCESS;
325 }

```



```

326  /* ----- */
327  /**      DisplayChangeFrame(DisplayCtrl *dispPtr, u32 frameIndex)
328  **      Parameters:
329  **          dispPtr - Pointer to the initialized DisplayCtrl struct
330  **          frameIndex - Index of the framebuffer to change to (must
331  **                      be between 0 and (DISPLAY_NUM_FRAMES - 1))
332  **      Return Value: int
333  **          XST_SUCCESS if successful, XST_FAILURE otherwise
334  **      Errors:
335  **      Description:
336  **          Changes the frame currently being displayed.
337  */
338
339  int DisplayChangeFrame(DisplayCtrl *dispPtr, u32 frameIndex)
340  {
341      int Status;
342
343      dispPtr->curFrame = frameIndex;
344      /*
345       * If currently running, then the DMA needs to be told to start reading from the desired frame
346       * at the end of the current frame
347       */
348      if (dispPtr->state == DISPLAY_RUNNING)
349      {
350          Status = XAxiVdma_StartParking(&(dispPtr->vdma), dispPtr->curFrame, XAXIVDMA_READ);
351          if (Status != XST_SUCCESS)
352          {
353              xil_printf("Cannot change frame, unable to start parking %d\r\n", Status);
354              return XST_FAILURE;
355          }
356      }
357
358      return XST_SUCCESS;
359  }
360
361  u32 DisplayClkCountCalc(u32 divide)
362  {
363      u32 output = 0;
364      u32 divCalc = 0;
365
366      divCalc = DisplayClkDivider(divide);
367      if (divCalc == ERR_CLKDIVIDER)
368          output = ERR_CLKCOUNTCALC;
369      else
370          output = (0xFFF & divCalc) | ((divCalc << 10) & 0x00C00000);
371      return output;
372  }
373
374  u32 DisplayClkDivider(u32 divide)
375  {
376      u32 output = 0;
377      u32 highTime = 0;
378      u32 lowTime = 0;
379
380      if ((divide < 1) || (divide > 128))
381          return ERR_CLKDIVIDER;
382
383      if (divide == 1)
384          return 0x1041;
385
386      highTime = divide / 2;
387      if (divide & 0b1) //if divide is odd
388      {
389          lowTime = highTime + 1;
390          output = 1 << CLK_BIT_WEDGE;
391      }

```

```

392     else
393     {
394         lowTime = highTime;
395     }
396
397     output |= 0x03F & lowTime;
398     output |= 0xFC0 & (highTime << 6);
399     return output;
400 }
401
402 u32 DisplayClkFindReg (ClkConfig *regValues, ClkMode *clkParams)
403 {
404     if ((clkParams->fbmult < 2) || clkParams->fbmult > 64 )
405         return 0;
406
407     regValues->clk0L = DisplayClkCountCalc(clkParams->clkdiv);
408     if (regValues->clk0L == ERR_CLKCOUNTCALC)
409         return 0;
410
411     regValues->clkFBL = DisplayClkCountCalc(clkParams->fbmult);
412     if (regValues->clkFBL == ERR_CLKCOUNTCALC)
413         return 0;
414
415     regValues->clkFBH_clk0H = 0;
416
417     regValues->divclk = DisplayClkDivider(clkParams->maindiv);
418     if (regValues->divclk == ERR_CLKDIVIDER)
419         return 0;
420
421     regValues->lockL = (u32) (lock_lookup[clkParams->fbmult - 1] & 0xFFFFFFFF);
422
423     regValues->fltr_lockH = (u32) ((lock_lookup[clkParams->fbmult - 1] >> 32) & 0x000000FF);
424     regValues->fltr_lockH |= ((filter_lookup_low[clkParams->fbmult - 1] << 16) & 0x03FF0000);
425
426     return 1;
427 }
428
429 void DisplayClkWriteReg (ClkConfig *regValues, u32 dispCtrlAddr)
430 {
431     Xil_Out32(dispCtrlAddr + OFST_DISPLAY_CLK_L, regValues->clk0L);
432     Xil_Out32(dispCtrlAddr + OFST_DISPLAY_FB_L, regValues->clkFBL);
433     Xil_Out32(dispCtrlAddr + OFST_DISPLAY_FB_H_CLK_H, regValues->clkFBH_clk0H);
434     Xil_Out32(dispCtrlAddr + OFST_DISPLAY_DIV, regValues->divclk);
435     Xil_Out32(dispCtrlAddr + OFST_DISPLAY_LOCK_L, regValues->lockL);
436     Xil_Out32(dispCtrlAddr + OFST_DISPLAY_FLTR_LOCK_H, regValues->fltr_lockH);
437 }
438
439 /*
440  * TODO: This function currently requires that the reference clock is 100MHz.
441  *       This should be changed so that the ref. clock can be specified, or read directly
442  *       out of hardware.
443  */
444 double DisplayClkFindParams(double freq, ClkMode *bestPick)
445 {
446     double bestError = 2000.0;
447     double curError;
448     double curClkMult;
449     double curFreq;
450     u32 curDiv, curFb, curClkDiv;
451     u32 minFb = 0;
452     u32 maxFb = 0;
453
454     bestPick->freq = 0.0;
455     /*
456     * TODO: replace with a smarter algorithm that doesn't check every possible combination
457     */

```

```

458     for (curDiv = 1; curDiv <= 10; curDiv++)
459     {
460         minFb = curDiv * 6; //This accounts for the 100MHz input and the 600MHz minimum VCO
461         maxFb = curDiv * 12; //This accounts for the 100MHz input and the 1200MHz maximum VCO
462         if (maxFb > 64)
463             maxFb = 64;
464         //This multiplier is used to find the best clkDiv value for each FB value
465         curClkMult = (100.0 / (double) curDiv) / freq;
466
467         curFb = minFb;
468         while (curFb <= maxFb)
469         {
470             curClkDiv = (u32) ((curClkMult * (double)curFb) + 0.5);
471             curFreq = ((100.0 / (double) curDiv) / (double) curClkDiv) * (double) curFb;
472             curError = fabs(curFreq - freq);
473             if (curError < bestError)
474             {
475                 bestError = curError;
476                 bestPick->clkdiv = curClkDiv;
477                 bestPick->fbmult = curFb;
478                 bestPick->maindiv = curDiv;
479                 bestPick->freq = curFreq;
480             }
481             curFb++;
482         }
483     }
484 }
485
486 return bestError;
487 }

```

Display_ctrl.h

```
1  /*****
2  /*
3  /* display_ctrl.h  --      Digilent Display Controller Driver
4  /*
5  /*****
6  /* Author: Sam Bobrowicz
7  /* Copyright 2014, Digilent Inc.
8  /*****
9  /* Module Description:
10 /*
11 /*      This module provides an easy to use API for controlling the
12 /*      Digilent display controller core (axi_dispctrl). It supports
13 /*      run-time resolution setting and seamless framebuffer-swapping
14 /*      for tear-free animation.
15 /*
16 /*      To use this driver, you must have an axi_dispctrl and axi_vdma
17 /*      core present in your system. For information on how to properly */
18 /*      configure these cores within your design, refer to the
19 /*      axi_dispctrl data sheet accessible from Vivado and XPS.
20 /*
21 /*      The following steps should be followed to use this driver:
22 /*      1) Create a DisplayCtrl object and pass a pointer to it to
23 /*      DisplayInitialize.
24 /*      2) Call DisplaySetMode to set the desired mode
25 /*      3) Call DisplayStart to begin outputting data to the display
26 /*      4) To create a seamless animation, draw the next image to a
27 /*      framebuffer currently not being displayed. Then call
28 /*      DisplayChangeFrame to begin displaying that frame.
29 /*      Repeat as needed, only ever modifying inactive frames.
30 /*      5) To change the resolution, call DisplaySetMode, followed by
31 /*      DisplayStart again.
32 /*
33 /*      This module contains code from the Xilinx Demo titled
34 /*      "xiicps_polled_master_example.c." Xilinx XAPP888 was also
35 /*      referenced for information on reconfiguring the MMCM or PLL.
36 /*      Note that functions beginning with "DisplayClk" are used
37 /*      internally for this purpose and should not need to be called
38 /*      externally.
39 /*
40 /*****
41 /* Revision History:
42 /*
43 /*      2/20/2014(SamB): Created
44 /*
45 /*****
46
47 #ifndef DISPLAY_CTRL_H_
48 #define DISPLAY_CTRL_H_
49 /* ----- */
50 /*      Include File Definitions
51 /* ----- */
52 #include "xil_types.h"
53 #include "vga_modes.h"
54 #include "xaxivdma.h"
55 /* ----- */
56 /*      Miscellaneous Declarations
57 /* ----- */
58 #define CLK_BIT_WEDGE 13
59 #define CLK_BIT_NOCOUNT 12
60
61 #define ERR_CLKCOUNTCALC 0xFFFFFFFF //This value is used to signal an error
62
63 #define OFST_DISPLAY_CTRL 0x0
64 #define OFST_DISPLAY_STATUS 0x4
```

```

65 #define OFST_DISPLAY_VIDEO_START 0x8
66 #define OFST_DISPLAY_CLK_L 0x1C
67 #define OFST_DISPLAY_FB_L 0x20
68 #define OFST_DISPLAY_FB_H_CLK_H 0x24
69 #define OFST_DISPLAY_DIV 0x28
70 #define OFST_DISPLAY_LOCK_L 0x2C
71 #define OFST_DISPLAY_FLTR_LOCK_H 0x30
72
73 #define BIT_DISPLAY_RED 16
74 #define BIT_DISPLAY_BLUE 0
75 #define BIT_DISPLAY_GREEN 8
76
77 #define BIT_DISPLAY_START 0
78 #define BIT_DISPLAY_RUNNING 1
79
80 #define DISPLAY_NOT_HDMI 0
81 #define DISPLAY_HDMI 1
82
83 /* This driver currently supports 3 frames. */
84 #define DISPLAY_NUM_FRAMES 3
85
86 /* WEDGE and NOCOUNT can't both be high, so this is used to signal an error state */
87 #define ERR_CLKDIVIDER (1 << CLK_BIT_WEDGE | 1 << CLK_BIT_NOCOUNT)
88 /* ----- */
89 /* General Type Declaration */
90 /* ----- */
91 typedef enum {
92     DISPLAY_STOPPED = 0,
93     DISPLAY_RUNNING = 1
94 } DisplayState;
95
96 typedef struct {
97     u32 clk0L;
98     u32 clkFBL;
99     u32 clkFBH_clk0H;
100     u32 divclk;
101     u32 lockL;
102     u32 fltr_lockH;
103 } ClkConfig;
104
105 typedef struct {
106     double freq;
107     u32 fbmult;
108     u32 clkdiv;
109     u32 maindiv;
110 } ClkMode;
111
112 typedef struct {
113     u32 dispCtrlAddr; /*Physical Base address of the disp_ctrl core*/
114     int fHdmi; /*flag indicating if the display controller is being used to drive an HDMI transmitter*/
115     XAxiVdma vdma; /*VDMA driver struct*/
116     XAxiVdma_DmaSetup vdmaConfig; /*VDMA channel configuration*/
117     VideoMode vMode; /*Current Video mode*/
118     u32 *framePtr[DISPLAY_NUM_FRAMES]; /* Array of pointers to the framebuffers */
119     u32 stride; /* The line stride of the framebuffers, in bytes */
120     double pxlFreq; /* Frequency of clock currently being generated */
121     u32 curFrame; /* Current frame being displayed */
122     DisplayState state; /* Indicates if the Display is currently running */
123 } DisplayCtrl;
124 /* ----- */
125 /* Variable Declarations */
126 /* ----- */
127 static const u64 lock_lookup[64] = {
128     0b0011000110111110100011111010010000000001,
129     0b0011000110111110100011111010010000000001,
130     0b0100001000111110100011111010010000000001,

```

```

131 0b01011010111111010001111101001000000001,
132 0b011100111011111010001111101001000000001,
133 0b100011000111111010001111101001000000001,
134 0b100111001111111010001111101001000000001,
135 0b101101011011111010001111101001000000001,
136 0b110011100111111010001111101001000000001,
137 0b111001110011111010001111101001000000001,
138 0b111111111111000010011111101001000000001,
139 0b111111111110011100111111101001000000001,
140 0b111111111110111011101111101001000000001,
141 0b111111111110101111001111101001000000001,
142 0b111111111110100010101111101001000000001,
143 0b111111111110011100011111101001000000001,
144 0b111111111110001111111111101001000000001,
145 0b111111111110001001101111101001000000001,
146 0b111111111110000011011111101001000000001,
147 0b11111111111011111010011111101001000000001,
148 0b1111111111101110110111111101001000000001,
149 0b1111111111101110000101111101001000000001,
150 0b1111111111101101010011111101001000000001,
151 0b11111111111011001000011111101001000000001,
152 0b11111111111011001000011111101001000000001,
153 0b11111111111010111011111111101001000000001,
154 0b11111111111010101111101111101001000000001,
155 0b11111111111010101111101111101001000000001,
156 0b1111111111101010001011111101001000000001,
157 0b1111111111101010001011111101001000000001,
158 0b11111111111010010110011111101001000000001,
159 0b11111111111010010110011111101001000000001,
160 0b11111111111010010110011111101001000000001,
161 0b111111111110100010011111101001000000001,
162 0b111111111110100010011111101001000000001,
163 0b111111111110100010011111101001000000001,
164 0b111111111110011110101111101001000000001,
165 0b111111111110011110101111101001000000001,
166 0b111111111110011110101111101001000000001,
167 0b111111111110011110101111101001000000001,
168 0b111111111110011110101111101001000000001,
169 0b111111111110011110101111101001000000001,
170 0b111111111110011110101111101001000000001,
171 0b111111111110011110101111101001000000001,
172 0b111111111110011110101111101001000000001,
173 0b111111111110011110101111101001000000001,
174 0b111111111110011110101111101001000000001,
175 0b111111111110011110101111101001000000001,
176 0b111111111110011110101111101001000000001,
177 0b111111111110011110101111101001000000001,
178 0b111111111110011110101111101001000000001,
179 0b111111111110011110101111101001000000001,
180 0b111111111110011110101111101001000000001,
181 0b111111111110011110101111101001000000001,
182 0b111111111110011110101111101001000000001,
183 0b111111111110011110101111101001000000001,
184 0b111111111110011110101111101001000000001,
185 0b111111111110011110101111101001000000001,
186 0b111111111110011110101111101001000000001,
187 0b111111111110011110101111101001000000001,
188 0b111111111110011110101111101001000000001,
189 0b111111111110011110101111101001000000001,
190 0b111111111110011110101111101001000000001,
191 0b111111111110011110101111101001000000001
192 };
193
194 static const u32 filter_lookup_low[64] = {
195     0b0001011111,
196     0b0001010111,

```

```

197     0b0001111011,
198     0b0001011011,
199     0b0001101011,
200     0b0001110011,
201     0b0001110011,
202     0b0001110011,
203     0b0001110011,
204     0b0001001011,
205     0b0001001011,
206     0b0001001011,
207     0b0010110011,
208     0b0001010011,
209     0b0001010011,
210     0b0001010011,
211     0b0001010011,
212     0b0001010011,
213     0b0001010011,
214     0b0001010011,
215     0b0001010011,
216     0b0001010011,
217     0b0001010011,
218     0b0001100011,
219     0b0001100011,
220     0b0001100011,
221     0b0001100011,
222     0b0001100011,
223     0b0001100011,
224     0b0001100011,
225     0b0001100011,
226     0b0001100011,
227     0b0001100011,
228     0b0001100011,
229     0b0001100011,
230     0b0001100011,
231     0b0001100011,
232     0b0010010011,
233     0b0010010011,
234     0b0010010011,
235     0b0010010011,
236     0b0010010011,
237     0b0010010011,
238     0b0010010011,
239     0b0010010011,
240     0b0010010011,
241     0b0010010011,
242     0b0010100011,
243     0b0010100011,
244     0b0010100011,
245     0b0010100011,
246     0b0010100011,
247     0b0010100011,
248     0b0010100011,
249     0b0010100011,
250     0b0010100011,
251     0b0010100011,
252     0b0010100011,
253     0b0010100011,
254     0b0010100011,
255     0b0010100011,
256     0b0010100011,
257     0b0010100011,
258     0b0010100011
259 };
260 /*-----*/
261 /*      Procedure Declarations      */
262 /*-----*/

```

```

263 u32 DisplayClkCountCalc(u32 divide);
264 u32 DisplayClkDivider(u32 divide);
265 u32 DisplayClkFindReg (ClkConfig *regValues, ClkMode *clkParams);
266 void DisplayClkWriteReg (ClkConfig *regValues, u32 dispCtrlAddr);
267 double DisplayClkFindParams(double freq, ClkMode *bestPick);
268
269 int DisplayStop(DisplayCtrl *dispPtr);
270 int DisplayStart(DisplayCtrl *dispPtr);
271 int DisplayInitialize(DisplayCtrl *dispPtr, u16 vdmaId, u32 dispCtrlAddr, int fHdmi, u32
    *framePtr[DISPLAY_NUM_FRAMES], u32 stride);
272 int DisplaySetMode(DisplayCtrl *dispPtr, const VideoMode *newMode);
273 int DisplayChangeFrame(DisplayCtrl *dispPtr, u32 frameIndex);
274
275 /* ----- */
276 /****** */
277 #endif /* DISPLAY_CTRL_H_ */

```


timer_ps.c

```
1  /*****
2  /*
3  /* timer_ps.c      --      Timer Delay    for Zynq systems
4  /*
5  /*****
6  /* Author: Sam Bobrowicz
7  /* Copyright 2014, Digilent Inc.
8  /*****
9  /* Module Description:
10 /*
11 /*      Implements an accurate delay function using the scutimer.
12 /*      Code from this module will cause conflicts with other code that */
13 /*      requires the Zynq's scu timer.
14 /*
15 /*      This module contains code from the Xilinx Demo titled
16 /*      "xscutimer_polled_example.c"
17 /*
18 /*****
19 /* Revision History:
20 /*
21 /*      2/14/2014(SamB): Created
22 /*
23 /*****
24
25
26 /* ----- */
27 /*      Include File Definitions
28 /* ----- */
29 #include "timer_ps.h"
30 #include "xscutimer.h"
31 #include "xil_types.h"
32
33 /* ----- */
34 /*      Global Variables
35 /* ----- */
36
37 XScuTimer TimerInstance; /* Cortex A9 Scu Private Timer Instance */
38
39 /* ----- */
40 /*      Procedure Definitions
41 /* ----- */
42
43 /***      TimerInitialize(u16 TimerDeviceId)
44 **      Parameters:
45 **          TimerDeviceId - The DEVICE ID of the Zynq SCU TIMER
46 **      Return Value: int
47 **          XST_SUCCESS if successful
48 **      Errors:
49 **      Description: Configures the global timer struct to access the
50 **                  the SCU timer. Can be called multiple times without error.
51 */
52 int TimerInitialize(u16 TimerDeviceId)
53 {
54     int Status;
55     XScuTimer *TimerInstancePtr = &TimerInstance;
56     XScuTimer_Config *ConfigTmrPtr;
57
58     /*
59      * Initialize the Scu Private Timer driver.
60      */
61     ConfigTmrPtr = XScuTimer_LookupConfig(TimerDeviceId);
62
63     /*
64      * This is where the virtual address would be used, this example
```

```

65     * uses physical address. Note that it is not considered an error
66     * if the timer has already been initialized.
67     */
68     Status = XScuTimer_CfgInitialize(TimerInstancePtr, ConfigTmrPtr,
69                                     ConfigTmrPtr->BaseAddr);
70     if (Status != XST_SUCCESS || Status != XST_DEVICE_IS_STARTED) {
71         return XST_FAILURE;
72     }
73
74     /*
75     * Set prescaler to 1
76     */
77     XScuTimer_SetPrescaler(TimerInstancePtr, 0);
78
79     return Status;
80 }
81 /* ----- */
82 /**      TimerDelay(u32 uSDelay)
83 **      Parameters:
84 **          uSDelay - Desired delay in micro seconds
85 **      Return Value:
86 **      Errors:
87 **      Description: Blocks execution for the desired amount of time.
88 **                  TimerInitialize must have been called at least once before calling this function.
89 **
90 /* ----- */
91 void TimerDelay(u32 uSDelay)
92 {
93     u32 timerCnt;
94
95     timerCnt = (TIMER_FREQ_HZ / 1000000) * uSDelay;
96
97     XScuTimer_Stop(&TimerInstance);
98     XScuTimer_DisableAutoReload(&TimerInstance);
99     XScuTimer_LoadTimer(&TimerInstance, timerCnt);
100    XScuTimer_Start(&TimerInstance);
101    while (XScuTimer_GetCounterValue(&TimerInstance))
102    {}
103
104    return;
105 }

```

timer_ps.h

```
1  /*******
2  /*
3  /* timer_ps.h      --      Timer Delay    for Zynq systems
4  /*
5  /*******
6  /* Author: Sam Bobrowicz
7  /* Copyright 2014, Digilent Inc.
8  /*******
9  /* Module Description:
10 /*
11 /*      Implements an accurate delay function using the scutimer.
12 /*      Code from this module will cause conflicts with other code that */
13 /*      requires the Zynq's scu timer.
14 /*
15 /*      This module contains code from the Xilinx Demo titled
16 /*      "xscutimer_polled_example.c"
17 /*
18 /*******
19 /* Revision History:
20 /*
21 /*      2/14/2014(SamB): Created
22 /*
23 /*******
24 #ifndef TIMER_PS_H_
25 #define TIMER_PS_H_
26
27 #include "xil_types.h"
28 #include "xparameters.h"
29
30 /* ----- */
31 /*      Miscellaneous Declarations
32 /* ----- */
33
34 #define TIMER_FREQ_HZ (XPAR_CPU_CORTEXA9_0_CPU_CLK_FREQ_HZ / 2)
35
36 /* ----- */
37 /*      Procedure Declarations
38 /* ----- */
39
40 int TimerInitialize(u16 TimerDeviceId);
41 void TimerDelay(u32 uSDelay);
42
43 /* ----- */
44
45 /*******
46
47 #endif /* TIMER_H_ */
```

vga_modes.h

```
1  /*******
2  /*
3  /* vga_modes.h    --    VideoMode definitions
4  /*
5  /*******
6  /* Author: Sam Bobrowicz
7  /* Copyright 2014, Digilent Inc.
8  /*******
9  /* Module Description:
10 /*
11 /*      This file contains the definition of the VideoMode type, and
12 /*      also defines several common video modes
13 /*
14 /*******
15 /* Revision History:
16 /*
17 /*      2/17/2014(SamB): Created
18 /*
19 /*******
20
21 #ifndef VGA_MODES_H_
22 #define VGA_MODES_H_
23
24 typedef struct {
25     char label[64]; /* Label describing the resolution */
26     u32 width; /*Width of the active video frame*/
27     u32 height; /*Height of the active video frame*/
28     u32 hps; /*Start time of Horizontal sync pulse, in pixel clocks (active width + H. front porch)*/
29     u32 hpe; /*End time of Horizontal sync pulse, in pixel clocks (active width + H. front porch + H. sync width)*/
30     u32 hmax; /*Total number of pixel clocks per line (active width + H. front porch + H. sync width + H. back porch) */
31     u32 hpol; /*hsync pulse polarity*/
32     u32 vps; /*Start time of Vertical sync pulse, in lines (active height + V. front porch)*/
33     u32 vpe; /*End time of Vertical sync pulse, in lines (active height + V. front porch + V. sync width)*/
34     u32 vmax; /*Total number of lines per frame (active height + V. front porch + V. sync width + V. back porch) */
35     u32 vpol; /*vsync pulse polarity*/
36     double freq; /*Pixel Clock frequency*/
37 } VideoMode;
38
39 static const VideoMode VMODE_640x480 = {
40     .label = "640x480@60Hz",
41     .width = 640,
42     .height = 480,
43     .hps = 656,
44     .hpe = 752,
45     .hmax = 799,
46     .hpol = 0,
47     .vps = 490,
48     .vpe = 492,
49     .vmax = 524,
50     .vpol = 0,
51     .freq = 25.0
52 };
53
54 static const VideoMode VMODE_800x600 = {
55     .label = "800x600@60Hz",
56     .width = 800,
57     .height = 600,
58     .hps = 840,
59     .hpe = 968,
60     .hmax = 1055,
61     .hpol = 1,
62     .vps = 601,
63     .vpe = 605,
64     .vmax = 627,
```

```

65     .vpol = 1,
66     .freq = 40.0
67 };
68
69 static const VideoMode VMODE_1280x1024 = {
70     .label = "1280x1024@60Hz",
71     .width = 1280,
72     .height = 1024,
73     .hps = 1328,
74     .hpe = 1440,
75     .hmax = 1687,
76     .hpol = 1,
77     .vps = 1025,
78     .vpe = 1028,
79     .vmax = 1065,
80     .vpol = 1,
81     .freq = 108.0
82 };
83
84 static const VideoMode VMODE_1280x720 = {
85     .label = "1280x720@60Hz",
86     .width = 1280,
87     .height = 720,
88     .hps = 1390,
89     .hpe = 1430,
90     .hmax = 1649,
91     .hpol = 1,
92     .vps = 725,
93     .vpe = 730,
94     .vmax = 749,
95     .vpol = 1,
96     .freq = 74.25, //74.2424 is close enough
97 };
98
99 static const VideoMode VMODE_1920x1080 = {
100     .label = "1920x1080@60Hz",
101     .width = 1920,
102     .height = 1080,
103     .hps = 2008,
104     .hpe = 2052,
105     .hmax = 2199,
106     .hpol = 1,
107     .vps = 1084,
108     .vpe = 1089,
109     .vmax = 1124,
110     .vpol = 1,
111     .freq = 148.5 //148.57 is close enough
112 };
113
114 #endif /* VGA_MODES_H_ */

```