

[Linux-fbdev-devel] [PATCH] atmel_lcdfb: AT91/AT32 LCD Controller framebuffer driver

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-05/msg04394.html>

- *From:* Nicolas Ferre <nicolas.ferre@xxxxxxxxxxxxxx>
- *Date:* Wed, 09 May 2007 16:59:46 +0200

From: Nicolas Ferre <nicolas.ferre@xxxxxxxxxxxxxx>

Adds a framebuffer driver to ATMEL AT91SAM9x and AT32 aka AVR32 platforms. Those chips share quite the same IP and this code is suitable for both architectures.

Signed-off-by: Nicolas Ferre <nicolas.ferre@xxxxxxxxxxxxxx>

This second patch is rebuild following Antonio A. Daplas comments. Thank you Tony.

The header file resides in an arch neutral directory.

AT91 platform informations are directly submitted trough the at91 maintainer.

```
drivers/video/Kconfig | 16 drivers/video/Makefile | 1 drivers/video/atmel_lcdfb.c | 752
+++++
include/video/atmel_lcd.h | 196 ++++++++
4 files changed, 965 insertions(+)
```

Index: b/include/video/atmel_lcd.h

```
=====
--- /dev/null
+++ b/include/video/atmel_lcd.h
@@ -0,0 +1,196 @@
+/*
+ * Header file for AT91/AT32 LCD Controller
+ *
+ * Data structure and register user interface
+ *
+ * Copyright (C) 2007 Atmel Corporation
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ */
```

```
+ * This program is distributed in the hope that it will be useful,  
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of  
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
+ * GNU General Public License for more details.  
+ *  
+ * You should have received a copy of the GNU General Public License  
+ * along with this program; if not, write to the Free Software  
+ * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA  
+ */  
+#ifndef __ATMEL_LCDC_H_  
+#define __ATMEL_LCDC_H_  
+  
+ /* LCD Controller info data structure */  
+struct atmel_lcdfb_info {  
+ spinlock_t lock;  
+ struct fb_info *info;  
+ void __iomem *mmio;  
+ unsigned long irq_base;  
+  
+ unsigned int guard_time;  
+ struct platform_device *pdev;  
+ struct clk *bus_clk;  
+ struct clk *lcdc_clk;  
+ unsigned int default_bpp;  
+ unsigned int default_lcdcon2;  
+ unsigned int default_dmacon;  
+ void (*atmel_lcdfb_power_control)(int on);  
+ struct fb_monspecs *default_monspecs;  
+ u32 pseudo_palette[16];  
+};  
+  
+#define ATMEL_LCDC_DMABADDR1 0x00  
+#define ATMEL_LCDC_DMABADDR2 0x04  
+#define ATMEL_LCDC_DMAFRMPT1 0x08  
+#define ATMEL_LCDC_DMAFRMPT2 0x0c  
+#define ATMEL_LCDC_DMAFRMADD1 0x10  
+#define ATMEL_LCDC_DMAFRMADD2 0x14  
+  
+#define ATMEL_LCDC_DMAFRMCFG 0x18  
+#define ATMEL_LCDC_FRSIZE (0x7ffff << 0)  
+#define ATMEL_LCDC_BLENGTH_OFFSET 24  
+#define ATMEL_LCDC_BLENGTH (0x7f << ATMEL_LCDC_BLENGTH_OFFSET)  
+  
+#define ATMEL_LCDC_DMACON 0x1c  
+#define ATMEL_LCDC_DMAEN (0x1 << 0)  
+#define ATMEL_LCDC_DMARST (0x1 << 1)  
+#define ATMEL_LCDC_DMABUSY (0x1 << 2)  
+#define ATMEL_LCDC_DMAUPDT (0x1 << 3)  
+#define ATMEL_LCDC_DMA2DEN (0x1 << 4)  
+  
+#define ATMEL_LCDC_DMA2DCFG 0x20
```

```
+#define ATMEL_LCDC_ADDRINC_OFFSET 0
+#define ATMEL_LCDC_ADDRINC (0xffff)
+#define ATMEL_LCDC_PIXELOFF_OFFSET 24
+#define ATMEL_LCDC_PIXELOFF (0x1f << 24)
+
+#define ATMEL_LCDC_LCDCON1 0x0800
+#define ATMEL_LCDC_BYPASS (1 << 0)
+#define ATMEL_LCDC_CLKVAL_OFFSET 12
+#define ATMEL_LCDC_CLKVAL (0x1ff << ATMEL_LCDC_CLKVAL_OFFSET)
+#define ATMEL_LCDC_LINCNT (0x7ff << 21)
+
+#define ATMEL_LCDC_LCDCON2 0x0804
+#define ATMEL_LCDC_DISTYPE (3 << 0)
+#define ATMEL_LCDC_DISTYPE_STNMONO (0 << 0)
+#define ATMEL_LCDC_DISTYPE_STNCOLOR (1 << 0)
+#define ATMEL_LCDC_DISTYPE_TFT (2 << 0)
+#define ATMEL_LCDC_SCANMOD (1 << 2)
+#define ATMEL_LCDC_SCANMOD_SINGLE (0 << 2)
+#define ATMEL_LCDC_SCANMOD_DUAL (1 << 2)
+#define ATMEL_LCDC_IFWIDTH (3 << 3)
+#define ATMEL_LCDC_IFWIDTH_4 (0 << 3)
+#define ATMEL_LCDC_IFWIDTH_8 (1 << 3)
+#define ATMEL_LCDC_IFWIDTH_16 (2 << 3)
+#define ATMEL_LCDC_PIXELSIZE (7 << 5)
+#define ATMEL_LCDC_PIXELSIZE_1 (0 << 5)
+#define ATMEL_LCDC_PIXELSIZE_2 (1 << 5)
+#define ATMEL_LCDC_PIXELSIZE_4 (2 << 5)
+#define ATMEL_LCDC_PIXELSIZE_8 (3 << 5)
+#define ATMEL_LCDC_PIXELSIZE_16 (4 << 5)
+#define ATMEL_LCDC_PIXELSIZE_24 (5 << 5)
+#define ATMEL_LCDC_PIXELSIZE_32 (6 << 5)
+#define ATMEL_LCDC_INVVD (1 << 8)
+#define ATMEL_LCDC_INVVD_NORMAL (0 << 8)
+#define ATMEL_LCDC_INVVD_INVERTED (1 << 8)
+#define ATMEL_LCDC_INVFRAME (1 << 9)
+#define ATMEL_LCDC_INVFRAME_NORMAL (0 << 9)
+#define ATMEL_LCDC_INVFRAME_INVERTED (1 << 9)
+#define ATMEL_LCDC_INVLIN (1 << 10)
+#define ATMEL_LCDC_INVLIN_NORMAL (0 << 10)
+#define ATMEL_LCDC_INVLIN_INVERTED (1 << 10)
+#define ATMEL_LCDC_INVCLK (1 << 11)
+#define ATMEL_LCDC_INVCLK_NORMAL (0 << 11)
+#define ATMEL_LCDC_INVCLK_INVERTED (1 << 11)
+#define ATMEL_LCDC_INVDVAL (1 << 12)
+#define ATMEL_LCDC_INVDVAL_NORMAL (0 << 12)
+#define ATMEL_LCDC_INVDVAL_INVERTED (1 << 12)
+#define ATMEL_LCDC_CLKMOD (1 << 15)
+#define ATMEL_LCDC_CLKMOD_ACTIVEDISPLAY (0 << 15)
+#define ATMEL_LCDC_CLKMOD_ALWAYSACTIVE (1 << 15)
+#define ATMEL_LCDC_MEMOR (1 << 31)
+#define ATMEL_LCDC_MEMOR_BIG (0 << 31)
```

```
+#define ATMEL_LCDC_MEMOR_LITTLE (1 << 31)
+
+#define ATMEL_LCDC_TIM1 0x0808
+#define ATMEL_LCDC_VFP (0xff << 0)
+#define ATMEL_LCDC_VBP_OFFSET 8
+#define ATMEL_LCDC_VBP (0xff << ATMEL_LCDC_VBP_OFFSET)
+#define ATMEL_LCDC_VPW_OFFSET 16
+#define ATMEL_LCDC_VPW (0x3f << ATMEL_LCDC_VPW_OFFSET)
+#define ATMEL_LCDC_VHDLY_OFFSET 24
+#define ATMEL_LCDC_VHDLY (0xf << ATMEL_LCDC_VHDLY_OFFSET)
+
+#define ATMEL_LCDC_TIM2 0x080c
+#define ATMEL_LCDC_HBP (0xff << 0)
+#define ATMEL_LCDC_HPW_OFFSET 8
+#define ATMEL_LCDC_HPW (0x3f << ATMEL_LCDC_HPW_OFFSET)
+#define ATMEL_LCDC_HFP_OFFSET 21
+#define ATMEL_LCDC_HFP (0x7ff << ATMEL_LCDC_HFP_OFFSET)
+
+#define ATMEL_LCDC_LCDFRMCFG 0x0810
+#define ATMEL_LCDC_LINEVAL (0x7ff << 0)
+#define ATMEL_LCDC_HOZVAL_OFFSET 21
+#define ATMEL_LCDC_HOZVAL (0x7ff << ATMEL_LCDC_HOZVAL_OFFSET)
+
+#define ATMEL_LCDC_FIFO 0x0814
+#define ATMEL_LCDC_FIFOTH (0xffff)
+
+#define ATMEL_LCDC_MVAL 0x0818
+
+#define ATMEL_LCDC_DP1_2 0x081c
+#define ATMEL_LCDC_DP4_7 0x0820
+#define ATMEL_LCDC_DP3_5 0x0824
+#define ATMEL_LCDC_DP2_3 0x0828
+#define ATMEL_LCDC_DP5_7 0x082c
+#define ATMEL_LCDC_DP3_4 0x0830
+#define ATMEL_LCDC_DP4_5 0x0834
+#define ATMEL_LCDC_DP6_7 0x0838
+#define ATMEL_LCDC_DP1_2_VAL (0xff)
+#define ATMEL_LCDC_DP4_7_VAL (0xffffffff)
+#define ATMEL_LCDC_DP3_5_VAL (0xffff)
+#define ATMEL_LCDC_DP2_3_VAL (0xfff)
+#define ATMEL_LCDC_DP5_7_VAL (0xffffffff)
+#define ATMEL_LCDC_DP3_4_VAL (0xffff)
+#define ATMEL_LCDC_DP4_5_VAL (0xffff)
+#define ATMEL_LCDC_DP6_7_VAL (0xffffffff)
+
+#define ATMEL_LCDC_PWRCON 0x083c
+#define ATMEL_LCDC_PWR (1 << 0)
+#define ATMEL_LCDC_GUARDT_OFFSET 1
+#define ATMEL_LCDC_GUARDT (0x7f << ATMEL_LCDC_GUARDT_OFFSET)
+#define ATMEL_LCDC_BUSY (1 << 31)
+
```

```
+#define ATMEL_LCDC_CONTRAST_CTR 0x0840
+#define ATMEL_LCDC_PS (3 << 0)
+#define ATMEL_LCDC_PS_DIV1 (0 << 0)
+#define ATMEL_LCDC_PS_DIV2 (1 << 0)
+#define ATMEL_LCDC_PS_DIV4 (2 << 0)
+#define ATMEL_LCDC_PS_DIV8 (3 << 0)
+#define ATMEL_LCDC_POL (1 << 2)
+#define ATMEL_LCDC_POL_NEGATIVE (0 << 2)
+#define ATMEL_LCDC_POL_POSITIVE (1 << 2)
+#define ATMEL_LCDC_ENA (1 << 3)
+#define ATMEL_LCDC_ENA_PWMDISABLE (0 << 3)
+#define ATMEL_LCDC_ENA_PWMENABLE (1 << 3)
+
+#define ATMEL_LCDC_CONTRAST_VAL 0x0844
+#define ATMEL_LCDC_CVAL (0xff)
+
+#define ATMEL_LCDC_IER 0x0848
+#define ATMEL_LCDC_IDR 0x084c
+#define ATMEL_LCDC_IMR 0x0850
+#define ATMEL_LCDC_ISR 0x0854
+#define ATMEL_LCDC_ICR 0x0858
+#define ATMEL_LCDC_LNI (1 << 0)
+#define ATMEL_LCDC_LSTLNI (1 << 1)
+#define ATMEL_LCDC_EOFI (1 << 2)
+#define ATMEL_LCDC_UFLWI (1 << 4)
+#define ATMEL_LCDC_OWRI (1 << 5)
+#define ATMEL_LCDC_MERI (1 << 6)
+
+#define ATMEL_LCDC_LUT(n) (0x0c00 + ((n)*4))
+
+#endif /* __ATMEL_LCDC_H__ */
Index: b/drivers/video/atmel_lcdfb.c
=====
--- /dev/null
+++ b/drivers/video/atmel_lcdfb.c
@@ -0,0 +1,752 @@
+/*
+ * Driver for AT91/AT32 LCD Controller
+ *
+ * Copyright (C) 2007 Atmel Corporation
+ *
+ * This file is subject to the terms and conditions of the GNU General Public
+ * License. See the file COPYING in the main directory of this archive for
+ * more details.
+ */
+
+#include <linux/kernel.h>
+#include <linux/platform_device.h>
+#include <linux/dma-mapping.h>
+#include <linux/interrupt.h>
+#include <linux/clk.h>
```

```

+#include <linux/fb.h>
+#include <linux/init.h>
+#include <linux/delay.h>
+
+#include <asm/arch/board.h>
+#include <asm/arch/cpu.h>
+#include <asm/arch/gpio.h>
+
+#include <video/atmel_lcdc.h>
+
+#define lcdc_readl(sinfo, reg) __raw_readl((sinfo)->mmio+(reg))
+#define lcdc_writel(sinfo, reg, val) __raw_writel((val), (sinfo)->mmio+(reg))
+
+/* configurable parameters */
+#define ATMEL_LCDC_CVAL_DEFAULT 0xc8
+#define ATMEL_LCDC_DMA_BURST_LEN 8
+
+#if defined(CONFIG_ARCH_AT91SAM9263)
+#define ATMEL_LCDC_FIFO_SIZE 2048
+#else
+#define ATMEL_LCDC_FIFO_SIZE 512
+#endif
+
+#if defined(CONFIG_ARCH_AT91)
+#define ATMEL_LCDFB_FBINFO_DEFAULT FBINFO_DEFAULT
+
+static inline void atmel_lcdfb_update_dma2d(struct atmel_lcdfb_info *sinfo,
+ struct fb_var_screeninfo *var)
+{
+
+}
+#elif defined(CONFIG_AVR32)
+#define ATMEL_LCDFB_FBINFO_DEFAULT (FBINFO_DEFAULT \
+ | FBINFO_PARTIAL_PAN_OK \
+ | FBINFO_HWACCEL_XPAN \
+ | FBINFO_HWACCEL_YPAN)
+
+static void atmel_lcdfb_update_dma2d(struct atmel_lcdfb_info *sinfo,
+ struct fb_var_screeninfo *var)
+{
+ u32 dma2dcfg;
+ u32 pixeloff;
+
+ pixeloff = (var->xoffset * var->bits_per_pixel) & 0x1f;
+
+ dma2dcfg = ((var->xres_virtual - var->xres) * var->bits_per_pixel) / 8;
+ dma2dcfg |= pixeloff << ATMEL_LCDC_PIXELOFF_OFFSET;
+ lcdc_writel(sinfo, ATMEL_LCDC_DMA2DCFG, dma2dcfg);
+
+ /* Update configuration */
+ lcdc_writel(sinfo, ATMEL_LCDC_DMACON,

```

```

+ lcdc_readl(sinfo, ATMEL_LCDC_DMACON)
+ | ATMEL_LCDC_DMAUPDT);
+}
+
+
+static struct fb_fix_screeninfo atmel_lcdfb_fix __initdata = {
+ .type = FB_TYPE_PACKED_PIXELS,
+ .visual = FB_VISUAL_TRUECOLOR,
+ .xpanstep = 0,
+ .ypanstep = 0,
+ .ywrapstep = 0,
+ .accel = FB_ACCEL_NONE,
+};
+
+
+static void atmel_lcdfb_update_dma(struct fb_info *info,
+ struct fb_var_screeninfo *var)
+{
+ struct atmel_lcdfb_info *sinfo = info->par;
+ struct fb_fix_screeninfo *fix = &info->fix;
+ unsigned long dma_addr;
+
+ dma_addr = (fix->smem_start + var->yoffset * fix->line_length
+ + var->xoffset * var->bits_per_pixel / 8);
+
+ dma_addr &= ~3UL;
+
+ /* Set framebuffer DMA base address and pixel offset */
+ lcdc_writel(sinfo, ATMEL_LCDC_DMABADDR1, dma_addr);
+
+ atmel_lcdfb_update_dma2d(sinfo, var);
+}
+
+static inline void atmel_lcdfb_free_video_memory(struct atmel_lcdfb_info *sinfo)
+{
+ struct fb_info *info = sinfo->info;
+
+ dma_free_writecombine(info->device, info->fix.smem_len,
+ info->screen_base, info->fix.smem_start);
+}
+
+/**
+ * atmel_lcdfb_alloc_video_memory - Allocate framebuffer memory
+ * @sinfo: the frame buffer to allocate memory for
+ */
+static int atmel_lcdfb_alloc_video_memory(struct atmel_lcdfb_info *sinfo)
+{
+ struct fb_info *info = sinfo->info;
+ struct fb_var_screeninfo *var = &info->var;
+
+

```

```

+ info->fix.smem_len = (var->xres_virtual * var->yres_virtual
+ * ((var->bits_per_pixel + 7) / 8));
+
+ info->screen_base = dma_alloc_writecombine(info->device, info->fix.smem_len,
+ (dma_addr_t *)&info->fix.smem_start, GFP_KERNEL);
+
+ if (!info->screen_base) {
+ return -ENOMEM;
+ }
+
+ return 0;
+}
+
+/**
+ * atmel_lcdfb_check_var - Validates a var passed in.
+ * @var: frame buffer variable screen structure
+ * @info: frame buffer structure that represents a single frame buffer
+ *
+ * Checks to see if the hardware supports the state requested by
+ * var passed in. This function does not alter the hardware
+ * state!!! This means the data stored in struct fb_info and
+ * struct atmel_lcdfb_info do not change. This includes the var
+ * inside of struct fb_info. Do NOT change these. This function
+ * can be called on its own if we intent to only test a mode and
+ * not actually set it. The stuff in modedb.c is a example of
+ * this. If the var passed in is slightly off by what the
+ * hardware can support then we alter the var PASSED in to what
+ * we can do. If the hardware doesn't support mode change a
+ * -EINVAL will be returned by the upper layers. You don't need
+ * to implement this function then. If you hardware doesn't
+ * support changing the resolution then this function is not
+ * needed. In this case the driver would just provide a var that
+ * represents the static state the screen is in.
+ *
+ * Returns negative errno on error, or zero on success.
+ */
+static int atmel_lcdfb_check_var(struct fb_var_screeninfo *var,
+ struct fb_info *info)
+{
+ struct device *dev = info->device;
+ struct atmel_lcdfb_info *sinfo = info->par;
+ unsigned long clk_value_khz;
+
+ clk_value_khz = clk_get_rate(sinfo->lcdc_clk) / 1000;
+
+ dev_dbg(dev, "%s:\n", __func__);
+ dev_dbg(dev, " resolution: %ux%u\n", var->xres, var->yres);
+ dev_dbg(dev, " pixclk: %lu KHz\n", PICOS2KHZ(var->pixclock));
+ dev_dbg(dev, " bpp: %u\n", var->bits_per_pixel);
+ dev_dbg(dev, " clk: %lu KHz\n", clk_value_khz);
+
+

```

```

+ if ((PICOS2KHZ(var->pixclock) * var->bits_per_pixel / 8) > clk_value_khz) {
+ dev_err(dev, "%lu KHz pixel clock is too fast\n", PICOS2KHZ(var->pixclock));
+ return -EINVAL;
+ }
+
+ /* Force same alignment for each line */
+ var->xres = (var->xres + 3) & ~3UL;
+ var->xres_virtual = (var->xres_virtual + 3) & ~3UL;
+
+ var->red.msb_right = var->green.msb_right = var->blue.msb_right = 0;
+ var->transp.msb_right = 0;
+ var->transp.offset = var->transp.length = 0;
+ var->xoffset = var->yoffset = 0;
+
+ switch (var->bits_per_pixel) {
+ case 2:
+ case 4:
+ case 8:
+ var->red.offset = var->green.offset = var->blue.offset = 0;
+ var->red.length = var->green.length = var->blue.length
+ = var->bits_per_pixel;
+ break;
+ case 15:
+ case 16:
+ var->red.offset = 0;
+ var->green.offset = 5;
+ var->blue.offset = 10;
+ var->red.length = var->green.length = var->blue.length = 5;
+ break;
+ case 24:
+ case 32:
+ var->red.offset = 0;
+ var->green.offset = 8;
+ var->blue.offset = 16;
+ var->red.length = var->green.length = var->blue.length = 8;
+ break;
+ default:
+ dev_err(dev, "color depth %d not supported\n",
+ var->bits_per_pixel);
+ return -EINVAL;
+ }
+
+ return 0;
+}
+
+/**
+ * atmel_lcdfb_set_par - Alters the hardware state.
+ * @info: frame buffer structure that represents a single frame buffer
+ *
+ * Using the fb_var_screeninfo in fb_info we set the resolution
+ * of the this particular framebuffer. This function alters the

```

```

+ * par AND the fb_fix_screeninfo stored in fb_info. It doesn't
+ * not alter var in fb_info since we are using that data. This
+ * means we depend on the data in var inside fb_info to be
+ * supported by the hardware. atmel_lcdfb_check_var is always called
+ * before atmel_lcdfb_set_par to ensure this. Again if you can't
+ * change the resolution you don't need this function.
+ *
+ */
+static int atmel_lcdfb_set_par(struct fb_info *info)
+{
+ struct atmel_lcdfb_info *sinfo = info->par;
+ unsigned long value;
+ unsigned long clk_value_khz;
+
+ dev_dbg(info->device, "%s:\n", __func__);
+ dev_dbg(info->device, " * resolution: %ux%u (%ux%u virtual)\n",
+ info->var.xres, info->var.yres,
+ info->var.xres_virtual, info->var.yres_virtual);
+
+ /* Turn off the LCD controller and the DMA controller */
+ lcdc_writel(sinfo, ATMEL_LCDC_PWRCON, sinfo->guard_time <<
+ ATMEL_LCDC_GUARDT_OFFSET);
+
+ lcdc_writel(sinfo, ATMEL_LCDC_DMACON, 0);
+
+ if (info->var.bits_per_pixel <= 8)
+ info->fix.visual = FB_VISUAL_PSEUDOCOLOR;
+ else
+ info->fix.visual = FB_VISUAL_TRUECOLOR;
+
+ info->fix.line_length = info->var.xres_virtual * (info->var.bits_per_pixel / 8);
+
+ /* Re-initialize the DMA engine... */
+ dev_dbg(info->device, " * update DMA engine\n");
+ atmel_lcdfb_update_dma(info, &info->var);
+
+ /* ...set frame size and burst length = 8 words (?) */
+ value = (info->var.yres * info->var.xres * info->var.bits_per_pixel) / 32;
+ value |= ((ATMEL_LCDC_DMA_BURST_LEN - 1) << ATMEL_LCDC_BLENGTH_OFFSET);
+ lcdc_writel(sinfo, ATMEL_LCDC_DMAFRMCFG, value);
+
+ /* Now, the LCDC core... */
+
+ /* Set pixel clock */
+ clk_value_khz = clk_get_rate(sinfo->lcdc_clk) / 1000;
+
+ value = clk_value_khz / PICOS2KHZ(info->var.pixclock);
+
+ if (clk_value_khz % PICOS2KHZ(info->var.pixclock))
+ value++;
+
+

```

```

+ value = (value / 2) - 1;
+
+ if (value <= 0) {
+ dev_notice(info->device, "Bypassing pixel clock divider\n");
+ lcdc_writel(sinfo, ATMEL_LCDC_LCDCON1, ATMEL_LCDC_BYPASS);
+ } else
+ lcdc_writel(sinfo, ATMEL_LCDC_LCDCON1, value << ATMEL_LCDC_CLKVAL_OFFSET);
+
+ /* Initialize control register 2 */
+ value = sinfo->default_lcdcon2;
+
+ if (!(info->var.sync & FB_SYNC_HOR_HIGH_ACT))
+ value |= ATMEL_LCDC_INVLINE_INVERTED;
+ if (!(info->var.sync & FB_SYNC_VERT_HIGH_ACT))
+ value |= ATMEL_LCDC_INVFRAME_INVERTED;
+
+ switch (info->var.bits_per_pixel) {
+ case 1: value |= ATMEL_LCDC_PIXELSIZE_1; break;
+ case 2: value |= ATMEL_LCDC_PIXELSIZE_2; break;
+ case 4: value |= ATMEL_LCDC_PIXELSIZE_4; break;
+ case 8: value |= ATMEL_LCDC_PIXELSIZE_8; break;
+ case 15: /* fall through */
+ case 16: value |= ATMEL_LCDC_PIXELSIZE_16; break;
+ case 24: value |= ATMEL_LCDC_PIXELSIZE_24; break;
+ case 32: value |= ATMEL_LCDC_PIXELSIZE_32; break;
+ default: BUG(); break;
+ }
+ dev_dbg(info->device, " * LCDCON2 = %08lx\n", value);
+ lcdc_writel(sinfo, ATMEL_LCDC_LCDCON2, value);
+
+ /* Vertical timing */
+ value = (info->var.vsync_len - 1) << ATMEL_LCDC_VPW_OFFSET;
+ value |= info->var.upper_margin << ATMEL_LCDC_VBP_OFFSET;
+ value |= info->var.lower_margin;
+ dev_dbg(info->device, " * LCDTIM1 = %08lx\n", value);
+ lcdc_writel(sinfo, ATMEL_LCDC_TIM1, value);
+
+ /* Horizontal timing */
+ value = (info->var.right_margin - 1) << ATMEL_LCDC_HFP_OFFSET;
+ value |= (info->var.hsync_len - 1) << ATMEL_LCDC_HPW_OFFSET;
+ value |= (info->var.left_margin - 1);
+ dev_dbg(info->device, " * LCDTIM2 = %08lx\n", value);
+ lcdc_writel(sinfo, ATMEL_LCDC_TIM2, value);
+
+ /* Display size */
+ value = (info->var.xres - 1) << ATMEL_LCDC_HOZVAL_OFFSET;
+ value |= info->var.yres - 1;
+ lcdc_writel(sinfo, ATMEL_LCDC_LCDFRMCFG, value);
+
+ /* FIFO Threshold: Use formula from data sheet */
+ value = ATMEL_LCDC_FIFO_SIZE - (2 * ATMEL_LCDC_DMA_BURST_LEN + 3);

```

```

+ lcdc_writel(sinfo, ATMEL_LCDC_FIFO, value);
+
+ /* Toggle LCD_MODE every frame */
+ lcdc_writel(sinfo, ATMEL_LCDC_MVAL, 0);
+
+ /* Disable all interrupts */
+ lcdc_writel(sinfo, ATMEL_LCDC_IDR, ~0UL);
+
+ /* Set contrast */
+ value = ATMEL_LCDC_PS_DIV8 | ATMEL_LCDC_POL_POSITIVE |
ATMEL_LCDC_ENA_PWMENABLE;
+ lcdc_writel(sinfo, ATMEL_LCDC_CONTRAST_CTR, value);
+ lcdc_writel(sinfo, ATMEL_LCDC_CONTRAST_VAL, ATMEL_LCDC_CVAL_DEFAULT);
+ /* ...wait for DMA engine to become idle... */
+ while (lcdc_readl(sinfo, ATMEL_LCDC_DMACON) & ATMEL_LCDC_DMABUSY)
+ msleep(10);
+
+ dev_dbg(info->device, " * re-enable DMA engine\n");
+ /* ...and enable it with updated configuration */
+ lcdc_writel(sinfo, ATMEL_LCDC_DMACON, sinfo->default_dmacon);
+
+ dev_dbg(info->device, " * re-enable LCDC core\n");
+ lcdc_writel(sinfo, ATMEL_LCDC_PWRCON,
+ (sinfo->guard_time << ATMEL_LCDC_GUARDT_OFFSET) | ATMEL_LCDC_PWR);
+
+ dev_dbg(info->device, " * DONE\n");
+
+ return 0;
+}
+
+static inline unsigned int chan_to_field(unsigned int chan, const struct fb_bitfield *bf)
+{
+ chan &= 0xffff;
+ chan >>= 16 - bf->length;
+ return chan << bf->offset;
+}
+
+/**
+ * atmel_lcdfb_setcolreg - Optional function. Sets a color register.
+ * @regno: Which register in the CLUT we are programming
+ * @red: The red value which can be up to 16 bits wide
+ * @green: The green value which can be up to 16 bits wide
+ * @blue: The blue value which can be up to 16 bits wide.
+ * @transp: If supported the alpha value which can be up to 16 bits wide.
+ * @info: frame buffer info structure
+ *
+ * Set a single color register. The values supplied have a 16 bit
+ * magnitude which needs to be scaled in this function for the hardware.
+ * Things to take into consideration are how many color registers, if
+ * any, are supported with the current color visual. With truecolor mode
+ * no color palettes are supported. Here a psuedo palette is created

```

```
+ * which we store the value in pseudo_palette in struct fb_info. For
+ * pseudocolor mode we have a limited color palette. To deal with this
+ * we can program what color is displayed for a particular pixel value.
+ * DirectColor is similar in that we can program each color field. If
+ * we have a static colormap we don't need to implement this function.
+ *
+ * Returns negative errno on error, or zero on success. In an
+ * ideal world, this would have been the case, but as it turns
+ * out, the other drivers return 1 on failure, so that's what
+ * we're going to do.
+ */
+static int atmel_lcdfb_setcolreg(unsigned int regno, unsigned int red,
+ unsigned int green, unsigned int blue,
+ unsigned int transp, struct fb_info *info)
+{
+ struct atmel_lcdfb_info *sinfo = info->par;
+ unsigned int val;
+ u32 *pal;
+ int ret = 1;
+
+ if (info->var.grayscale)
+ red = green = blue = (19595 * red + 38470 * green
+ + 7471 * blue) >> 16;
+
+ switch (info->fix.visual) {
+ case FB_VISUAL_TRUECOLOR:
+ if (regno < 16) {
+ pal = info->pseudo_palette;
+
+ val = chan_to_field(red, &info->var.red);
+ val |= chan_to_field(green, &info->var.green);
+ val |= chan_to_field(blue, &info->var.blue);
+
+ pal[regno] = val;
+ ret = 0;
+ }
+ break;
+
+ case FB_VISUAL_PSEUDOCOLOR:
+ if (regno < 256) {
+ val = ((red >> 11) & 0x001f);
+ val |= ((green >> 6) & 0x03e0);
+ val |= ((blue >> 1) & 0x7c00);
+
+ /*
+ * TODO: intensity bit. Maybe something like
+ * ~(red[10] ^ green[10] ^ blue[10]) & 1
+ */
+
+ lcdc_writel(sinfo, ATMEL_LCDC_LUT(regno), val);
+ ret = 0;
+ }
```

```
+ }
+ break;
+ }
+
+ return ret;
+}
+
+static int atmel_lcdfb_pan_display(struct fb_var_screeninfo *var,
+ struct fb_info *info)
+{
+ dev_dbg(info->device, "%s\n", __func__);
+
+ atmel_lcdfb_update_dma(info, var);
+
+ return 0;
+}
+
+static struct fb_ops atmel_lcdfb_ops = {
+ .owner = THIS_MODULE,
+ .fb_check_var = atmel_lcdfb_check_var,
+ .fb_set_par = atmel_lcdfb_set_par,
+ .fb_setcolreg = atmel_lcdfb_setcolreg,
+ .fb_pan_display = atmel_lcdfb_pan_display,
+ .fb_fillrect = cfb_fillrect,
+ .fb_copyarea = cfb_copyarea,
+ .fb_imageblit = cfb_imageblit,
+};
+
+static irqreturn_t atmel_lcdfb_interrupt(int irq, void *dev_id)
+{
+ struct fb_info *info = dev_id;
+ struct atmel_lcdfb_info *sinfo = info->par;
+ u32 status;
+
+ status = lcdc_readl(sinfo, ATMEL_LCDC_ISR);
+ lcdc_writel(sinfo, ATMEL_LCDC_IDR, status);
+ return IRQ_HANDLED;
+}
+
+static int __init atmel_lcdfb_init_fbinfo(struct atmel_lcdfb_info *sinfo)
+{
+ struct fb_info *info = sinfo->info;
+ int ret = 0;
+
+ memset_io(info->screen_base, 0, info->fix.smem_len);
+ info->var.activate |= FB_ACTIVATE_FORCE | FB_ACTIVATE_NOW;
+
+ dev_info(info->device,
+ "%luKiB frame buffer at %08lx (mapped at %p)\n",
+ (unsigned long)info->fix.smem_len / 1024,
+ (unsigned long)info->fix.smem_start,
```

```

+ info->screen_base);
+
+ /* Allocate colormap */
+ ret = fb_alloc_cmap(&info->cmap, 256, 0);
+ if (ret < 0)
+ dev_err(info->device, "Alloc color map failed\n");
+
+ return ret;
+}
+
+static void atmel_lcdfb_start_clock(struct atmel_lcdfb_info *sinfo)
+{
+ if (sinfo->bus_clk)
+ clk_enable(sinfo->bus_clk);
+ clk_enable(sinfo->lcdc_clk);
+}
+
+static void atmel_lcdfb_stop_clock(struct atmel_lcdfb_info *sinfo)
+{
+ if (sinfo->bus_clk)
+ clk_disable(sinfo->bus_clk);
+ clk_disable(sinfo->lcdc_clk);
+}
+
+static int __init atmel_lcdfb_probe(struct platform_device *pdev)
+{
+ struct device *dev = &pdev->dev;
+ struct fb_info *info;
+ struct atmel_lcdfb_info *sinfo;
+ struct atmel_lcdfb_info *pdata_sinfo;
+ struct resource *regs = NULL;
+ struct resource *map = NULL;
+ int ret;
+
+ dev_dbg(dev, "%s BEGIN\n", __func__);
+
+ ret = -ENOMEM;
+ info = framebuffer_alloc(sizeof(struct atmel_lcdfb_info), dev);
+ if (!info) {
+ dev_err(dev, "cannot allocate memory\n");
+ goto out;
+ }
+
+ sinfo = info->par;
+
+ if (dev->platform_data) {
+ pdata_sinfo = (struct atmel_lcdfb_info *)dev->platform_data;
+ sinfo->default_bpp = pdata_sinfo->default_bpp;
+ sinfo->default_dmacon = pdata_sinfo->default_dmacon;
+ sinfo->default_lcdcon2 = pdata_sinfo->default_lcdcon2;

```

```

+ sinfo->default_monspecs = pdata_sinfo->default_monspecs;
+ sinfo->atmel_lcdfb_power_control = pdata_sinfo->atmel_lcdfb_power_control;
+ sinfo->guard_time = pdata_sinfo->guard_time;
+ } else {
+ dev_err(dev, "cannot get default configuration\n");
+ goto free_info;
+ }
+ sinfo->info = info;
+ sinfo->pdev = pdev;
+
+ strcpy(info->fix.id, sinfo->pdev->name);
+ info->flags = ATMEL_LCDFB_FBINFO_DEFAULT;
+ info->pseudo_palette = sinfo->pseudo_palette;
+ info->fbops = &atmel_lcdfb_ops;
+
+ memcpy(&info->monspecs, sinfo->default_monspecs, sizeof(info->monspecs));
+ info->fix = atmel_lcdfb_fix;
+
+ /* Enable LCDC Clocks */
+ if (cpu_is_at91sam9261() || cpu_is_at32ap7000()) {
+ sinfo->bus_clk = clk_get(dev, "hck1");
+ if (IS_ERR(sinfo->bus_clk)) {
+ ret = PTR_ERR(sinfo->bus_clk);
+ goto free_info;
+ }
+ }
+ sinfo->lcdc_clk = clk_get(dev, "lcdc_clk");
+ if (IS_ERR(sinfo->lcdc_clk)) {
+ ret = PTR_ERR(sinfo->lcdc_clk);
+ goto put_bus_clk;
+ }
+ atmel_lcdfb_start_clock(sinfo);
+
+ ret = fb_find_mode(&info->var, info, NULL, info->monspecs.modedb,
+ info->monspecs.modedb_len, info->monspecs.modedb,
+ sinfo->default_bpp);
+ if (!ret) {
+ dev_err(dev, "no suitable video mode found\n");
+ goto stop_clk;
+ }
+
+
+ regs = platform_get_resource(pdev, IORESOURCE_MEM, 0);
+ if (!regs) {
+ dev_err(dev, "resources unusable\n");
+ ret = -ENXIO;
+ goto stop_clk;
+ }
+
+ sinfo->irq_base = platform_get_irq(pdev, 0);
+ if (sinfo->irq_base < 0) {

```

```

+ dev_err(dev, "unable to get irq\n");
+ ret = sinfo->irq_base;
+ goto stop_clk;
+ }
+
+ /* Initialize video memory */
+ map = platform_get_resource(pdev, IORESOURCE_MEM, 1);
+ if (map) {
+ /* use a pre-allocated memory buffer */
+ info->fix.smem_start = map->start;
+ info->fix.smem_len = map->end - map->start + 1;
+ if (!request_mem_region(info->fix.smem_start,
+ info->fix.smem_len, pdev->name)) {
+ ret = -EBUSY;
+ goto stop_clk;
+ }
+
+ info->screen_base = ioremap(info->fix.smem_start, info->fix.smem_len);
+ if (!info->screen_base)
+ goto release_intmem;
+ } else {
+ /* allocate memory buffer */
+ ret = atmel_lcdfb_alloc_video_memory(sinfo);
+ if (ret < 0) {
+ dev_err(dev, "cannot allocate framebuffer: %d\n", ret);
+ goto stop_clk;
+ }
+ }
+
+ /* LCDC registers */
+ info->fix.mmio_start = regs->start;
+ info->fix.mmio_len = regs->end - regs->start + 1;
+
+ if (!request_mem_region(info->fix.mmio_start,
+ info->fix.mmio_len, pdev->name)) {
+ ret = -EBUSY;
+ goto free_fb;
+ }
+
+ sinfo->mmio = ioremap(info->fix.mmio_start, info->fix.mmio_len);
+ if (!sinfo->mmio) {
+ dev_err(dev, "cannot map LCDC registers\n");
+ goto release_mem;
+ }
+
+ /* interrupt */
+ ret = request_irq(sinfo->irq_base, atmel_lcdfb_interrupt, 0, pdev->name, info);
+ if (ret) {
+ dev_err(dev, "request_irq failed: %d\n", ret);
+ goto unmap_mmio;
+ }

```

```
+
+ ret = atmel_lcdfb_init_fbinfo(sinfo);
+ if (ret < 0) {
+ dev_err(dev, "init fbinfo failed: %d\n", ret);
+ goto unregister_irqs;
+ }
+
+ /*
+ * This makes sure that our colour bitfield
+ * descriptors are correctly initialised.
+ */
+ atmel_lcdfb_check_var(&info->var, info);
+
+ ret = fb_set_var(info, &info->var);
+ if (ret) {
+ dev_warn(dev, "unable to set display parameters\n");
+ goto free_cmap;
+ }
+
+ dev_set_drvdata(dev, info);
+
+ /*
+ * Tell the world that we're ready to go
+ */
+ ret = register_framebuffer(info);
+ if (ret < 0) {
+ dev_err(dev, "failed to register framebuffer device: %d\n", ret);
+ goto free_cmap;
+ }
+
+ /* Power up the LCDC screen */
+ if (sinfo->atmel_lcdfb_power_control)
+ sinfo->atmel_lcdfb_power_control(1);
+
+ dev_info(dev, "fb%d: Atmel LCDC at 0x%08lx (mapped at %p), irq %lu\n",
+ info->node, info->fix.mmio_start, sinfo->mmio, sinfo->irq_base);
+
+ return 0;
+
+
+
+free_cmap:
+ fb_dealloc_cmap(&info->cmap);
+unregister_irqs:
+ free_irq(sinfo->irq_base, info);
+unmap_mmio:
+ iounmap(sinfo->mmio);
+release_mem:
+ release_mem_region(info->fix.mmio_start, info->fix.mmio_len);
+free_fb:
+ if (map)
+ iounmap(info->screen_base);
```

```
+ else
+ atmel_lcdfb_free_video_memory(sinfo);
+
+release_intmem:
+ if (map)
+ release_mem_region(info->fix.smem_start, info->fix.smem_len);
+stop_clk:
+ atmel_lcdfb_stop_clock(sinfo);
+ clk_put(sinfo->lcdc_clk);
+put_bus_clk:
+ if (sinfo->bus_clk)
+ clk_put(sinfo->bus_clk);
+free_info:
+ framebuffer_release(info);
+out:
+ dev_dbg(dev, "%s FAILED\n", __func__);
+ return ret;
+}
+
+static int __exit atmel_lcdfb_remove(struct platform_device *pdev)
+{
+ struct device *dev = &pdev->dev;
+ struct fb_info *info = dev_get_drvdata(dev);
+ struct atmel_lcdfb_info *sinfo = info->par;
+
+ if (!sinfo)
+ return 0;
+
+ if (sinfo->atmel_lcdfb_power_control)
+ sinfo->atmel_lcdfb_power_control(0);
+ unregister_framebuffer(info);
+ atmel_lcdfb_stop_clock(sinfo);
+ clk_put(sinfo->lcdc_clk);
+ if (sinfo->bus_clk)
+ clk_put(sinfo->bus_clk);
+ fb_dealloc_cmap(&info->cmap);
+ free_irq(sinfo->irq_base, info);
+ iounmap(sinfo->mmio);
+ release_mem_region(info->fix.mmio_start, info->fix.mmio_len);
+ if (platform_get_resource(pdev, IORESOURCE_MEM, 1)) {
+ iounmap(info->screen_base);
+ release_mem_region(info->fix.smem_start, info->fix.smem_len);
+ } else {
+ atmel_lcdfb_free_video_memory(sinfo);
+ }
+
+ dev_set_drvdata(dev, NULL);
+ framebuffer_release(info);
+
+ return 0;
+}
```

```
+
+static struct platform_driver atmel_lcdfb_driver = {
+ .remove = __exit_p(atmel_lcdfb_remove),
+ .driver = {
+ .name = "atmel_lcdfb",
+ .owner = THIS_MODULE,
+ },
+};
+
+static int __init atmel_lcdfb_init(void)
+{
+ return platform_driver_probe(&atmel_lcdfb_driver, atmel_lcdfb_probe);
+}
+
+static void __exit atmel_lcdfb_exit(void)
+{
+ platform_driver_unregister(&atmel_lcdfb_driver);
+}
+
+module_init(atmel_lcdfb_init);
+module_exit(atmel_lcdfb_exit);
+
+MODULE_DESCRIPTION("AT91/AT32 LCD Controller framebuffer driver");
+MODULE_AUTHOR("Nicolas Ferre <nicolas.ferre@xxxxxxxxxxxxxx>");
+MODULE_LICENSE("GPL");
```

Index: b/drivers/video/Kconfig

=====

```
--- a/drivers/video/Kconfig
+++ b/drivers/video/Kconfig
@@ -674,6 +674,22 @@
working with S1D13806). Product specs at
<http://www.erd.epson.com/vdc/html/legacy\_13xxx.htm>
```

```
+config FB_ATMEL
+ tristate "AT91/AT32 LCD Controller support"
+ depends on FB && (ARCH_AT91SAM9261 || ARCH_AT91SAM9263 || AVR32)
+ select FB_CFB_FILLRECT
+ select FB_CFB_COPYAREA
+ select FB_CFB_IMAGEBLIT
+ help
+ This enables support for the AT91/AT32 LCD Controller.
+
+config FB_INTSRAM
+ bool "Frame Buffer in internal SRAM"
+ depends on FB_ATMEL && ARCH_AT91SAM9261
+ help
+ Say Y if you want to map Frame Buffer in internal SRAM. Say N if you want
+ to let frame buffer in external SDRAM.
+
config FB_NVIDIA
tristate "nVidia Framebuffer Support"
```

depends on FB && PCI

Index: b/drivers/video/Makefile

```
=====  
--- a/drivers/video/Makefile  
+++ b/drivers/video/Makefile  
@@ -75,6 +75,7 @@  
obj-$(CONFIG_FB_SA1100) += sa1100fb.o  
obj-$(CONFIG_FB_HIT) += hitfb.o  
obj-$(CONFIG_FB_EPSON1355) += epson1355fb.o  
+obj-$(CONFIG_FB_ATMEL) += atmel_lcdfb.o  
obj-$(CONFIG_FB_PVR2) += pvr2fb.o  
obj-$(CONFIG_FB_VOODOO1) += sstfb.o  
obj-$(CONFIG_FB_ARMCLCD) += amba-clcd.o
```

—
To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@xxxxxxxxxxxxxxxxxxx
More majordomo info at <http://vger.kernel.org/majordomo-info.html>
Please read the FAQ at <http://www.tux.org/lkml/>