

文章编号:1007-2780(2010)01-0094-05

基于 ARM 和 FPGA 的视频监控系统设计

赵小欢,夏靖波,李明辉

(空军工程大学 电讯工程学院, 陕西 西安 710077, E-mail: zhaoxiaohuan0906@163.com)

摘 要: 设计并实现了一种基于 ARM+FPGA 的视频监控系统,以 ARM9 处理器为主控制器,FPGA 为协处理器,构造 ARM 与 FPGA 间的高速数据传输通道和基于 Linux 的轻量级的图形驱动,完成实时视频采集和显示。实验结果表明,该设计能够流畅播放 PAL/NTSC 两种制式视频信号,具有良好的扩展性、稳定性和较快的响应速度。

关 键 词: 视频监控;总线接口;图形驱动

中图分类号: TP311 **文献标识码:** A

Design of Video Monitoring System Based on ARM and FPGA

ZHAO Xiao-huan, XIA Jing-bo, LI Ming-hui

(Department of Telecommunication Engineering, Air Force Engineering University,
Xi'an 710077, China, E-mail: zhaoxiaohuan0906@163.com)

Abstract: A video monitoring system is designed and implemented based on ARM and FPGA. With ARM9 processor as master controller and FPGA as coprocessor, the functions of real-time video acquisition and display are realized on the basis of constructing high-speed data transmission channel between ARM and FPGA and establishing lightweight graphics drivers on Linux OS. The experiment shows that the system can play video fluently which are PAL or NTSC formats with the advantages of excellent expansibility, high stability and fast response speed.

Key words: video monitor; bus interface; graphics drivers

1 引 言

在工业生产控制中,最直接最有效的监控方法莫过于视频监控,其中 ARM 处理器由于具有较好的性能与较低的价格已经在各种工业控制领域广泛使用。但是,随着视频监控对分辨率和实时性的要求越来越高,普通的 ARM 处理器已难以满足需求,而高档的 ARM 处理器由于价格昂贵而难以普遍使用。采用多处理器架构是缓解性能与价格矛盾的一种较好的方案,但是在多处理器结构中,多个处理器之间的数据传输效率直接

制约着整个系统的性能,稍有不慎就会成为系统性能的瓶颈。

本文采用 ARM 与 FPGA 互连的主从结构来完成视频信号的采集:以 ARM 为系统的核心,协调整个系统的工作;以 FPGA 为协处理器连接在 ARM 的一个 BANK 上,完成前端视频信号的处理^[1]。该方案充分发挥了 ARM 与 FPGA 各自的优势,很好地实现了视频监控功能,系统具有很好的灵活性,稍加改造就能应用于其他的嵌入式设计中。

2 系统总体设计

视频信号十分复杂。复合视频信号将色差信号 B-Y 及 R-Y 以 3.58 MHz 的次载波载送在辉度信号 Y 信号里,同时还包括了行同步信号、行消隐信号、场同步信号、场消隐信号等同步信号。为了得到图像信号,首先需要采用梳状滤波器将 3.58 MHz 的色度信号与 Y 信号分离,得到 Y、C 信号(即 S-video 信号),然后以不同的速率对这两路信号采样、编码就得到各种标准的图像信号。采用分离的模拟器件来完成视频信号的滤波、抽样、编码等不仅麻烦,而且可靠性难以保证。

设计中采用 PHILIPS 公司的 SAA7111A 来完成复合视频信号的解码。SAA7111A 是一种增强型视频输入处理器芯片,能够完成视频制式自动识别、A/D 变换、同步时序信号分离、亮度色度调整、色彩空间变换等功能,极大地减轻了设计的复杂度和体积。

SAA7111A 解码出来的视频信号的分辨率为 720×576,同时包括场同步信号、行同步信号、奇偶场信号等同步信号。为了便于显示,需要将有效的视频信号缓存起来。此操作数据量大,但运算相对简单,特别适合具有并行处理能力的 FPGA 来实现。

ARM 具有优良的界面,支持操作系统,网络功能优越,具有丰富的外围设备,非常适合作系统的主控模块。

于显示,此处控制逻辑由 FPGA 完成。本系统采集到的是无损数据格式的图像,数据量十分巨大,当需要存储采集数据时,可将短时间内图像数据写入 U 盘或 SD 卡中用于 PC 机后期处理。

3 EP1C6Q240C8 与 AT91RM9200 的接口设计

ARM 与 FPGA 之间的数据传输速率对整个系统的性能至关重要,因此有必要设计一种高速可靠的传输方式。ARM 与 FPGA 主要有以下几种数据交换方式:

(1)FPGA 的 I/O 口与 ARM 的 GPIO 相连。此种方式最为简单,但数据传输速率极为有限,不适合高速视频数据。

(2)FPGA 与 ARM 通过大容量 FIFO 或双端口 RAM 完成通信。此种方式比较常见,且数据传输速率有保证,但也存在一些不足,缓存容量显得比较有限,同时大容量缓存价格也比较贵。

(3)FPGA 做为存储器连接在 ARM 的 BANK 上。此种方式较为灵活且传输速率高,将大容量的 SRAM 做为缓存连到 FPGA 上,通过 FPGA 编程将缓存中数据按照 ARM 总线时序输出,ARM 仅连续读数据即可。

显然采用方案 3 比较有优势。图 2 显示了 ARM 与 FPGA 的连接方式。为了保证通信的快速和稳定性,AT91RM9200 采用 EBI 外部总线接口和 FPGA 连接,将 FPGA 挂在 ARM9 的 BANK4 上^[2]。由于每次传送 16 位数据,需要将 ARM9 的 A[18:1]与 FPGA 相连,同时可以通过 FPGA 控制 ARM9 的 nwait 信号延长总线读写周期确保 ARM9 可靠读写。AT91RM9200 的 BANK4 的基地址是 0x400000000,这样 ARM 每次循环只需从 0x400000000 开始读一帧数据即可。

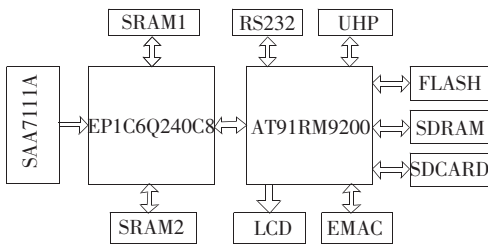


图 1 系统框图
Fig. 1 Structure of the system

视频采集系统的框图如图 1 所示,采用 Atmel 公司的 AT91RM9200 为整个系统的主控模块,FPGA 为协处理器负责分时缓存视频数据。AT91RM9200 采用 Linux 操作系统,当视频监控功能启动时,AT91RM9200 连续从 SRAM1 与 SRAM2 读取 FPGA 采集到的视频送入 LCD 用

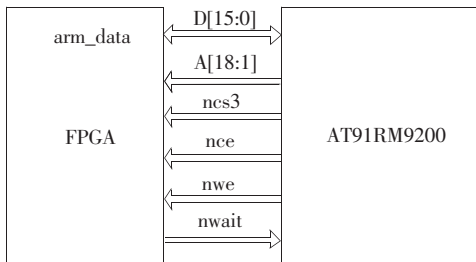


图 2 ARM 和 FPGA 接口图
Fig. 2 Interface between ARM and FPGA

为了确保 FPGA 不对 ARM 的正常工作造成干扰,需要在 ARM 不处于读 FPGA 的状态时,使 FPGA 和 ARM 的数据线隔离。具体的实现方法是:将 FPGA 的 ncs3 和 noe 置为输入引脚,当 $ncs3=0 \ \& \ noe=0$ 时, $arm_data[15:0]=sram_data[15:0]$,其他任何情况下 $arm_data[15:0]=16'bZZZZZZZZZZZZZZZZ$ 。

4 FPGA 前端程序设计

4.1 视频图像采集

系统中通过 AT91RM9200 的两线接口 TWI 对 SAA7111A 进行配置,通过 CCD 输入 PAL 制式的 CVBS 视频信号,输出 RGB(5:6:5)格式的数字视频信号,同时输出场同步信号 VREF、行同步信号 HREF、奇偶场信号 RTS0、像素时钟信号 LLC2,时序如图 3 所示。

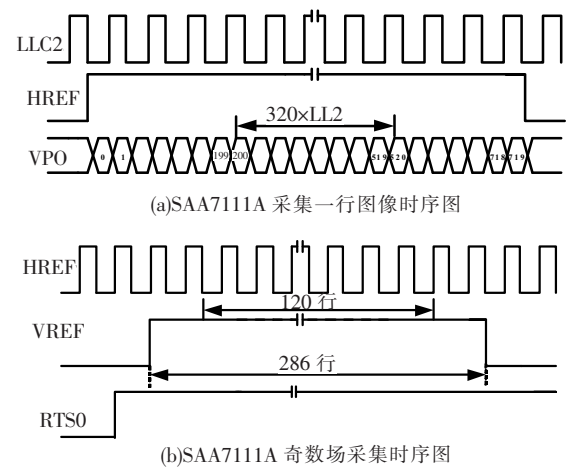


图 3 FPGA 图像采集时序图
Fig. 3 Image acquisition timing of FPGA

图像采集过程如下:当 $RTS0=1$ 时,表明输出为奇数场,当场同步信号 VREF 上升沿到来时,对行同步信号 HREF 计数到 83 时,启动行有效信号;同理,当行同步信号 HREF 上升沿到来时,对像素时钟信号 LLC2 计数到 200 时,启动数据有效信号,开始采集图像数据。采用该方式可以灵活控制采集信号的分辨率,这里将分辨率设为 320×240 。由于 SAA7111A 解码出来的数字视频信号采用隔行扫描的方式,所以该奇数场采集了 120 行数据。

对图像数据的定位采用下列方法:调用 FPGA 的 ROM 宏模块,建立一个表,保存行首地址,每

次存储数据时,行首地址采用查表得到,行内地址采用计数的方式^[3]。这样做可以提高系统的可靠性,即使某一个数据受到干扰,失步也只存在于该像素所在的行,其他的行仍然同步。

4.2 双缓存模块

由于系统中 FPGA 采集视频与 ARM 读取视频都是连续不断的,因此采用双缓存结构。双缓存结构原理如图 4 所示:数据流通过“输入数据流选择单元”,等时分配到两个数据缓冲模块;与此同时,轮流将两个数据缓冲模块的数据通过“输出数据流选择单元”送到“数据流运算处理模块”被运算处理^[4]。

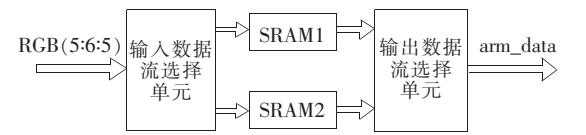


图 4 乒乓缓存结构
Fig. 4 Ping-pong cache structure

系统中选用的 SRAM 为 16 位数据位的 IS61LV25616AL,它的真值表如表 1 所示。通过 FPGA 合理控制 SRAM 的各个管脚来实现对双缓存的读写。

表 1 SRAM 真值表				
Table 1 Truth table of SRAM				
模式	$\overline{we}$	$\overline{ce}$	$\overline{oe}$	sram_data
不选择芯片工作	X	H	X	高阻
输出禁能	H	L	H	高阻
读	H	L	L	DOUT
写	L	L	L	DIN

系统中通过 Verilog HDL 编程实现对双缓存的设计。首先对 $RTS0$ 二分频得到 myrts 信号,当采集奇数帧视频图像时 myrts 为 1,偶数帧时为 0。当 myrts = 1 时,需要对 SRAM1 写操作,而对 SRAM2 读操作,此时, $sram1_noe=0$, $sram1_nwe=0$, $sram1_data=RGB[5:6:5]$,通过 $sram1_nce$ 来控制 SRAM1 写操作;而 $sram2_noe=arm_noe$, $sram2_nwe=1$, $sram2_nce=arm_ncs3$, $sram2_data=16'bZZZZZZZZZZZZZZZZ$,输出数据流 arm_data,当 ARM 总线处于读状态时, $arm_data=sram2_data$ 。而当 myrts = 0 时,

SRAM1 处于读状态,SRAM2 处于写状态,SRAM1 与 SRAM2 操作正好互换。双缓存切换通过下面代码实现^[5]:

```
assign sram1_noe=myrts? 1'b0:arm_noe;
assign sram2_noe=myrts? arm_noe:1'b0;
assign sram1_nwe=myrts? 1'b0:1'b1;
assign sram2_nwe=myrts? 1'b1:1'b0;
assign sram1_data=wsram1_en? RGB[5:6:
5]: 16'bZZZZZZZZZZZZZZZZ;
assign sram2_data=wsram2_en? RGB[5:6:
5]: 16'bZZZZZZZZZZZZZZZZ;
assign arm_data = myrts? (rsram2_en?
sram2_data:16'bZZZZZZZZZZZZZZZZ):(rsram1_en?
sram1_data:16'bZZZZZZZZZZZZZZZZ);
assign sram1_nce=myrts? sram1_nce_buf;
arm_ncs3;
assign sram2_nce=myrts? arm_ncs3:sram2_
nce_buf;
```

5 ARM 图形驱动程序开发

系统中使用的 AT91RM9200 是一款基于 ARM920T 内核的高性价比、低功耗、32 位的 ARM 芯片,集成了 EBI、PMC、Ethernet、USB、MCI、SSC、UASRT、SPI、RTC、TWI 等接口及其控制器,却没有针对 LCD 显示的控制器,所以本系统添加了 S1D13506 做为显示控制器。S1D13506 是 EPSON 大规模显示控制器家族中较新的一款。它内置 RAMDAC 硬件 X-Y 轴转置二维加速器,共有 114 个寄存器,可以灵活地对各种不同的显示方式进行设置,功能非常强大,可以和目前市场上流行的多种 CPU 总线兼容。

Linux 操作系统为 LCD 等显示设备提供了帧缓冲区 Framebuffer^[6]。Framebuffer 是 Linux 控制台下的一个通用的图形接口,它拥有良好的平台无关性,可以支持绝大多数的硬件,一些流行的图形库如 QT、GTK 便是基于 Framebuffer 的。它的接口非常简单,通过它可以十分方便地构建一个图形系统。

但是,本系统中主要强调的是显示的实时性,对图形库并没有过多的要求,此时 Linux 自带的 Framebuffer 显得过于臃肿,因此,这里自编了一个轻量级的图形驱动程序。下面将具体介绍该驱动程序编写方法。

5.1 模块初始化

Linux 下的驱动程序总是先调用 module_init(),因此初始化通过 module_init(fpga_init)来实现^[7]。

在初始化中使用 I/O 内存时存在以下问题:设备对应于某些约定的物理地址,但是 CPU 并没有预先定义访问它们的虚拟地址,因此必须有一种把虚拟地址赋予设备的方法。这里通过 void* ioremap_nocache()来实现物理内存地址和虚拟地址的映射,具体实现如下:

```
p13506Reg = (unsigned char*) ioremap_
nocache (S1D13506_REG_BASE, S1D13506_
REG_BASE_SIZE);
p13506Mem= (unsigned short*) ioremap_
nocache (S1D13506_MEM_BASE, S1D13506_
MEM_BASE_SIZE);
pFpgaBase = (unsigned short*) ioremap_
nocache (FPGA_BASE,FPGA_VEDIO_SIZE);
```

通过 ioremap_nocache 的映射,设备驱动程序能够使用任何 I/O 内存地址,从而完成设备的初始化。同时在 module_init()中还需通过 register_chrdev(FPGA_MAJOR, "fpga", &fpga_fops)完成设备文件向系统的注册。

5.2 填充 file_operations 结构

由于用户进程是通过设备文件同硬件打交道的,对设备文件的操作方式是一些系统调用,所以需要填充 file_operations 数据结构把系统调用和驱动程序关联起来,实现如下:

```
static struct file_operations fpga_fops =
{
    owner:      THIS_MODULE,
    ioctl:      fpga_ioctl,
    open:       fpga_open,
    release::    fpga_release,
};
```

该结构的每一个成员名字都对应着一个系统调用,用户进程利用系统调用对设备文件进行诸如 open 操作时,系统调用通过设备文件的主设备号找到相应的设备驱动程序,然后读取这个数据结构相应的函数指针,然后把控制权交给该函数。

5.3 图形驱动设备中操作的实现

上面设备驱动程序中定义了 fpga_ioctl、fpga_open 及 fpga_release 3 种方法,其中 fpga_ioctl 是

驱动中最重要的一个方法,它提供了更新窗口、界面切换、播放视频、暂停播放视频监控的大多数功能^[8],具体实现如图 5 所示。fpga_ioctl 的函数原型为:

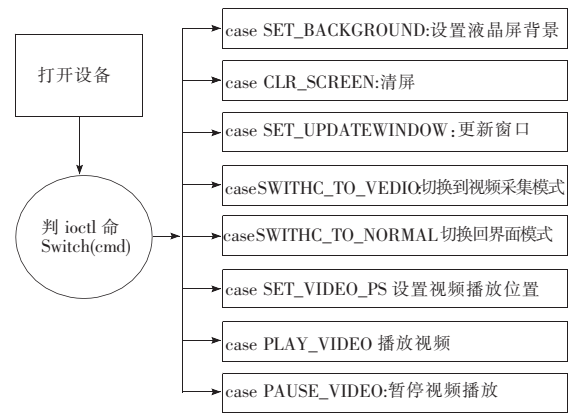


图 5 fpga_ioctl 方法
Fig. 5 Fpga_ioctl method

```
static int fpga_ioctl(struct inode* inode, struct
file* filp, unsigned int cmd, unsigned long arg)
{ };
```

6 结 论

随着 FPGA 的速度和对复杂数据处理能力的加强,ARM 对控制和网络功能的进一步完善,基于 ARM+FPGA 的双核系统将在未来有更广阔的应用前景。

本文设计的基于 ARM+FPGA 视频监视系统,很好地解决了多核数据传输的瓶颈,ARM 与 FPGA 之间数据传输的速率为 $320 \times 240 \times 16 \times 25 = 30.72$ Mbit/s,能够满足实时视频的需要;同时,通过编写轻量级的图形驱动,减轻了 ARM 主处理器的负担,使得 ARM 处理器能够同时处理其他任务。目前,本系统已成功应用到某铁路监控系统中,运行结果良好。

参 考 文 献:

[1] 高富强,刘国华,何斌,等. 基于 ARM9 与 FPGA 的工业 CT 数据传输系统的接口设计 [J]. 核电子学与探测技术, 2008,28(5):934-936.

[2] 胡小龙,周俊明,夏显忠,等. 飞机座舱图形显示加速系统设计及 FPGA 实现 [J]. 中南大学学报(自然科学版), 2008,39(5):1042-1048.

[3] 单洁,曹剑中,章奇,等. 高分辨率航拍相机实时图像存储系统的关键技术 [J]. 激光与红外,2006,36(3):234-237.

[4] 康艳霞,曹剑中,田雁,等. 实时视频处理系统中乒乓缓存的设计 [J]. 弹箭与制导学报,2007,24(4):218-221.

[5] 商广辉,代永平,黄茜,等. 场序彩色视频控制系统 [J]. 液晶与显示,2007,22(2):186-191.

[6] 雷鸣,熊文龙,杨单. 基于 Framebuffer 的 LCD 驱动程序的实现 [J]. 武汉理工大学学报(交通科学与工程版), 2006,30(1):106-109.

[7] Corbet J, Rubini A, Kroah-hertman G. *Linux Device Drivers* [M]. Cambridge:O'Reilly Media,2005:1-2.

[8] 金锋,武志辉. 轻量级嵌入式 Linux 图形驱动程序与图形用户接口 [J]. 北京理工大学学报,2008,28(3):233-236.