VxWorks 中低速串行设备驱动的层次化设计方法

曾 勇
（中国西南电子技术研究所，四川 成都 610036）

摘 要：针对传统开发模式存在的问题，在原有工程基础上对低速串行设备驱动程序进行了更改、封装，并提出了一种低速串行设备驱动层次化设计方法。

关键词：层次化设计方法；低速串行设备；设备驱动

中图分类号：TP368.2 **文献标识码**：A **文章编号**：1674-7720(2013)03-0073-03

Implementation of low-speed serial device driver hierarchical design method in VxWorks

Zeng Yong
(Southwest China Institute of Electronic Technology, Chengdu 610036, China)

Abstract: Making a simple discuss of traditional device driver development. In the original project based on low-speed serial device driver were changed, packaging, and on this basis put forward a low-speed serial device driver hierarchical design method.

Key words: hierarchical design method; low-speed serial device; device driver

随着通信技术的发展，低速串行设备动态扩展和多低速串行设备全双工通信技术在嵌入式领域的应用越来越广泛。利用现场可编程门阵列 FPGA（Field Programmable Gates Army）完全可以将若干接口电路的功能集成到一片 FPGA 中，这不仅具有集成度高、体积小和功耗低等优点，而且还具有用户可编程能力，同时还可实现整个系统的功能重构以及项目过程中一些特殊低速串行设备传输方式，这是专用低速串行设备实现芯片所不具备的功能。

嵌入式系统一般采用 PowerPC+FPGA 的架构方式，FPGA 主要完成底层接口协议处理，PowerPC 的任务主要实现控制功能。FPGA 与 PowerPC 通过内部总线（Local Bus）方式连接，FPGA 实现基于总线的时序操作；采用内存映射方式，FPGA 实现的多个低速串行设备的设置、操作寄存器以及 FIFO 的入口出口映射为 PowerPC 可寻址的一段内存空间，根据系统设计方法采用中断复用方式或者使用多个 Po werPC 中断号。

在驱动程序传统开发过程中，FPGA 实现的多个低速串行设备驱动程序的设计不受驱动体系约束，只是简单地根据上层应用的需求提供接口，导致应用程序代码可移植性、可读性差，开发流程并行度不高，层次性不明确。实际调试过程中则需要应用程序人员、驱动开发人员同时对各自的程序进行修改、重新编译。

本文在原有工程基础上对 FPGA 驱动程序进行了更改、封装，实现了基于 FPGA 的多个低速串行设备在 VxWorks 系统的标准 I/O 设备驱动，并在此基础上提出了一种低速串行设备驱动层次化设计方法。

1 低速串行设备驱动程序设计

在 VxWorks 中，串行设备是一种特殊字符设备。与字符设备不同的是，串行设备的驱动程序并不是直接挂在 I/O 系统中，而是通过虚拟设备 ttyDrv 来使用[1]。ttyDrv 与 I/O 系统以及真实驱动程序之间的关系如图 1 所示。

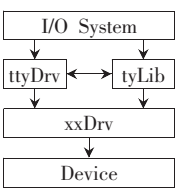

图 1 ttyDrv 与 I/O 系统以及真实驱动程序关系

ttyDrv 和 tyLib 实现了以下的功能[2]：(1)I/O 系统请求（在驱动表中增加入口函数，创建设备描述符等）；(2) 管理 I/O 系统的入口函数，如 tyOpen，ttyIoctl，tyRead，tyWrite 等；(3)管理 selectLib 的调用；(4)管理数据缓冲区以及缓冲区上的线程同步互斥；(5)ttyDrv 处理 open 和 ioctl 操作（ttyOpen and ttyIoctl）；(6)tyLib 处

理 read 和write 操作(tyRead and tyWrite)。

真实的低速串行设备驱动函数(xxDrv)通过 ttyDrv 安装的回调函数在 I/O 系统和设备之间移动数据;tyLib 提供了 2 个回调函数(tyITx 和 tyIRd)。

1.1 标准驱动程序的构造

SIO_DRV_FUNCS 结构体是串行设备驱动程序的一个重要的组成部分,它是驱动函数的入口,在串行设备的驱动程序中必须要有一个包含指向 SIO_DRV_FUNCS 指针的结构(xx_CHAN),xx_CHAN 包含了 xxDrv 中所需要的设备特定信息,以及提供给高层函数的底层操作函数;由于系统不知道驱动函数中所使用的数据结构,VxWorks 提供了一个数据结构 SIO_CHAN 来进行数据类型转换[3]。SIO_CHAN 结构仅定义了一个指向SIO_DRV_FUNCS 结构体的指针。

FPGA 实现的多个低速串行设备驱动程序的结构构造实例如下:

```
typedef struct{
  /*SIO_CHAN*MUST*be first*/
  SIO_CHAN sio;      /*standard SIO_CHAN element*/
  UINT32* inBase;              /*FIFO 入口地址 */
  UINT32* outBase;             /*FIFO 出口地址 */
  UINT32* baudFreqSetBase;   /* 波特率设置地址 */
  UINT32* IntSourceBase;     /* 中断源寄存器地址 */
  UINT32* inNumBase;
            /*FIFO 有效数据长度寄存器地址 */
  SEM_ID RecvSem;            /* 对应的信号量 */
  UINT32 intLevel;
  UINT32 IntSourceMask;    /* 中断源寄存器掩码 */
  /*callbacks*/
  STATUS (*getTxChar) (void*, char*);
  void (*putRcvChar) (void*, char);
  void * getTxArg;
  void * putRcvArg;
  } FPGA_CHAN;
```

1.2 驱动函数和回调函数的挂接

ttyDrv 通过 SIO_DRV_FUNCS 提供的入口对 xxDrv 的服务进行访问,xxDrv 通过回调函数来访问 ttyDrv 提供的服务[4]。xxDrv 驱动函数在 SIO_DRV_FUNCS 中的挂接实例如下:

```
LOCAL SIO_DRV_FUNCS fpgaSioDrvFuncs=
  {
  fpgaIoctl,      /*Support device specific ioctl cmds*/
  fpgaTxStartup,          /*Initiates a transmit cycle*/
  fpgaCallbackInstall,
        /*Installs access to higher level protocols */
  fpgaPollInput,          /*Poll mode input routine*/
  fpgaPollOutput          /*Poll mode output routine*/
  };
```

1.3 驱动程序标准操作函数的实现

1.3.1 标准写函数

标准的 I/O 系统写函数 write()调用 tyWrite()函数——ttyDrv 加载在驱动表中的写入口函数,tyWrite()函数将数据放入环形缓冲队列中并调用驱动函数 fpgaTxStartup()发起一次传输过程。write 函数的调用关系如图 2 所示。

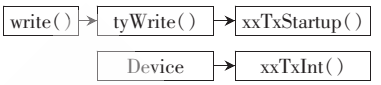

图 2 write 函数的调用关系

1.3.2 标准读函数

通过中断方式从 FPGA 生成的 FIFO 出口地址读取数据,通过绑定的回调函数将数据传送到接收环形缓冲队列中,并调用 ttyDrv 加载在驱动表中的读入口函数 tyRead()驱动函数,通知阻塞的 I/O 系统写函数 read()函数完成一次传输过程。read 函数的调用关系如图 3 所示。

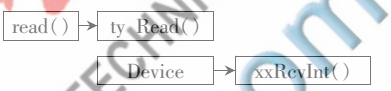

图 3 read 函数的调用关系

1.3.3 I/O 操作函数

标准的 I/O 系统写函数 ioctl()调用 ttyIoctl()函数,ttyIoctl()函数调用 FPGAIoctl()进行操作,根据系统的实际需要实现部分设置项。I/O 操作函数的调用关系如图 4 所示。

图 4 I/O 操作函数调用关系

2 低速串行设备的加载

(1)根据实际需求修改 config.h 中 NUM_TTYS 宏;

(2)参照 sysSerialHwInit()函数实现 FpgaSerialHwInit(),对 NUM_TTYS 个低速串行设备结构体 FPGA_CHAN 进行初始化以及驱动函数和回调函数的挂接;

(3)参照 sysSerialHwInit2()函数实现 FpgaSerialHwInit2(),挂接中断处理函数 fpgaRcvInt();

(4)调用 ttyDevCreate()函数创建 tty 设备,添加设备驱动表,生成环形数据缓冲区,安装回调函数 tyITx()和 tyIRd()。

3 低速串行设备驱动层次化设计方法

在不同的系统设计中,PowerPC 和 FPGA 硬件连接方式不尽相同,如不同的片选信号、总线上数据线高低位的不同接法,中断管脚接法的不同,导致 FPGA 实现的寄存器映射地址及使用的中断号存在差异;针对不同的应用程序和应用场所,外部接口对应的设备也会发生相应的变化。以上这些都会破坏程序的独立性、设备无关性。需要对驱动程序、应用程序进行重新编译、烧写,不利于版本的管理以及外场设备的维护。这是在设备驱动设计中需要考虑的一个问题。

由此本文提出了一种层次化的低速串行设备驱动

设计方法。将驱动程序的实现分为三层，硬件分离层、设备加载层和功能描述层，各自实现的功能如下：

（1）硬件分离层：通过 XML 文件实现低速串行驱动程序结构初始化和硬件设备实现差异的分离；

（2）设备加载层：实现低速串行设备在 VxWorks 系统的标准 I/O 设备驱动的加载；

（3）功能描述层：通过 XML 文件实现应用程序和低速串行设备之间的映射关系以及串行设备功能的描述。

低速串行设备驱动层次化设计方法流程如图 5 所示。

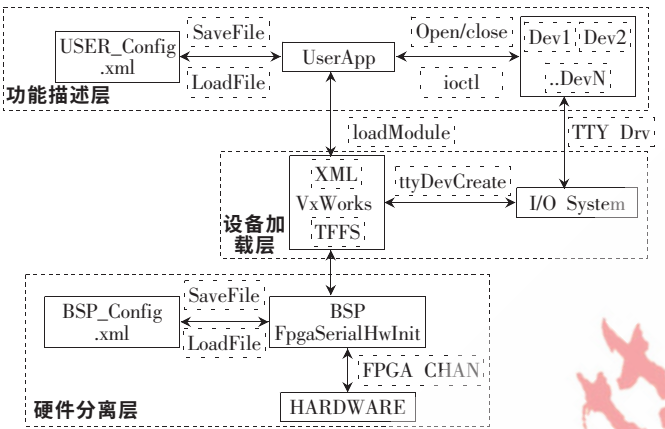

图 5 低速串行设备驱动层次化设计方法流程

3.1 层次化设计驱动程序的实现

嵌入式操作系统中，通常都使用 Flash 作为主存介质。在 Flash 上建立 TFFS 文件系统，建立文件系统后，类似于在 Windows 操作系统下对硬盘操作一样，进行数据的拷贝、删除以及文件的建立等操作。XML 作为配置文件的主要优点是改变参数配置时不需要改变和重新编译应用程序，只需在 XML 文件中更改就可以了。TinyXML 是一个简单小巧、且很容易集成到其他程序中的 C++XML 解析器。即 TinyXML 解析一个 XML 文档并由此生成一个可读、可修改、可保存的文档对象模型（DOM）。本文的实现过程中使用 TinyXML 对 XML 文档进行解析。

低速串行设备驱动层次化设计方法建立在文件系统之上，VxWorks 可以动态地加载用户程序，并在系统中加入对 XML 文件进行解析和存储的功能。

硬件分离层：FpgaSerialHwInit（）通过读取设备对应的 BSP_Config.xml 文件，对多个低速串行设备结构体 FPGA_CHAN 中的各个寄存器地址进行初始化以及驱动函数和回调函数的挂接。

设备加载层：实现标准 I/O 设备驱动的加载。

功能描述层：应用程序通过读取 USER_Config.xml 文件，获取设备的对外连接状态及设备提供的功能，对多个低速串行设备 read、write 以及 ioctl 等操作。

通常设备驱动人员只需要进行对 BSP_Config.xml 和 USER_Config.xml 两个配置文件进行修改、更新。

3.2 XML 配置文件的设计及实现

BSP_Config.xml 文件格式示例如下：

```
const char*bspXmlCfg=
"<?xml version=\"1.0\" encoding=\"UTF-8\" ?>"
"<Settings>"
"<DevSerial>"
"<inBase>\"10xd0100000\"</inBase>"
"<outBase>\"10xd0100100\"</outBase>"
……
"<IntSourceMask>\"10xd0100800\"</IntSourceMask>"
"</DevSerial>"
……
"</Settings>"
```

USER_Config.xml 文件格式示例如下：

```
const char*userXmlCfg=
"<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
"<Settings>"
    "<DevSerial>"
    "<DevSerial Config Port=\"2\" Baud=\"38400\"
                                    Enabled=\"1\"/>"
    "</DevSerial>"
……
"</Settings>"
```

XML 文件的遍历和读取方法如下：

```
TiXmlDocument doc(CONFIG_FILE_NAME);
bool loadOkay=doc.LoadFile();          /* 读取文件 */
doc.Print();                           /* 输出打印 */
rootNode=doc.FirstChild("Settings");   /* 获取根节点 */
……
node=rootNode->FirstChild("System");
                                       /* 获取匹配的子节点 */
itemElement=node->ToElement();         /* 转换为元素 */
itemElement->Attribute("Type",&dValue); /* 读取属性*/
```

XML 文件的保存和更新方法如下：

```
TiXmlDocument doc;
doc.Parse(demoXmlCfg);
                /* 读取需要保存的 XML 文件框架结构 */
rootNode=doc.FirstChild("Settings");   /* 获取根节点 */
……
node=rootNode->FirstChild("System");
                                       /* 获取匹配的子节点 */
itemElement=node->ToElement();         /* 转换为元素 */
itemElement->SetDoubleAttribute("Type",Type)
                                       ;/* 更新属性 */
……
doc.SaveFile(CONFIG_FILE_NAME);
                                       /* 存储更新配置文件 */
```

本文提出并实现了一种 VxWorks 低速串行设备驱动

的层次化设计方法，通过驱动程序功能分层设计方式，屏蔽各层的实现细节，克服了嵌入式系统开发过程中，由于设备驱动程序的设计不受驱动体系约束带来的程序代码的可移植性、可读性差，开发流程并行度不高，层次性不明确等不足之处；在实际的工程实现中节约了开发时间，极大地提高了效率。

参考文献

[1] Vxworks kernel programmers guide [Z].Wind River Systems Inc，2006.

[2] Vxworks device driver developers guide [Z]. Wind River Systems Inc，2006.

[3] 内核示例源代码[Z]. Wind River Systems Inc，2006.

[4] 陈智育，等.VxWorks 程序开发实践[M].北京：人民邮电出版社，2004.

（收稿日期：2012-11-29）

作者简介：

曾勇，男，1981 年生，硕士研究生，主要研究方向：嵌入式系统设计。