数字电视机顶盒应用软件系统中状态机的设计

乔英东　肖创柏
（北京工业大学计算机学院　北京 100022）

摘　要　数字电视机顶盒软件系统各模块的交互关系很复杂，容易造成机顶盒软件系统出错和不稳定。利用有限状态机模型，状态机作为各个进程的管理者可以确保机顶盒软件系统的稳定性和可靠性。

关键词　数字电视，机顶盒，状态机

The State-Machine Design for Digital TV Applaction Software System

QIAO Ying-Dong　XIAO Chuang-Bai
(School of Computer, Beijing University of Technology, Beijing 100022)

Abstract　The entity of STB software system is a compound of several separate processes, each has its unique function. This causes the state machine implementation much more complicated. State machine mechanism implements here as a top leader to ensure the stability and reliability of the STB software system.

Keywords　Digital TV, Set-top-box, State-machine

1　引言

随着技术的进步，电视已开始步入数字电视时代。数字电视的设备也越趋于先进，早期的只有简单电视接收功能的第一代数字电视机顶盒逐渐正在被第二代有着交互功能、上网等功能的数字电视机顶盒所代替。数字电视代替模拟电视、数字电视机代替模拟电视机是必然的发展趋势，但由于目前的模拟电视机的数量庞大，如果要把模拟电视机全部淘汰，换成数字电视机的话，那是一笔巨大的浪费。为此，数字电视机顶盒成为目前模拟电视向数字电视过渡的一个有效设备。机顶盒本身需要具有解压缩和模拟电视信号编码的能力。它负责把数字化的信号在用户终端转化成模拟信号输入给模拟电视机。

通过数字电视机顶盒，除了实现数字电视的基本功能即收看电视外，还可以扩展出许多增值服务，比如接收股票行情、视频点播、远程交互式教育等。还可以采用 IPOverDVB 技术为用户提供因特网接入服务。这就使得机顶盒软件系统各模块的交互关系很复杂，容易造成机顶盒软件系统出错和不稳定。状态机的机制是状态机作为各个进程的管理者来确保机顶盒软件系统的稳定性和可靠性。

在机顶盒应用系统的设计过程中，系统应用功能划分为一个个相对独立的任务之后，任务的组织方式主要取决于任务之间的数据传递，即任务之间的逻辑关系。每个任务都是通过程序的执行来改变周围环境、设备的状态，因此使用有限状态机(FSM: Finate State Machine)进行任务内部逻辑的分析、设计和开发是一个理想的选择。

2　有限状态机理论

许多嵌入式系统，特别是大型控制系统，整个分析机制与系统的状态密切相关，采用有限状态机对理解、分析和设计复杂系统很有帮助。

有限状态机是具有离散输入和输出系统的一种数学模型，它在任何时刻，都处于一个特定的状态。对于事件驱动的程序设计，它是非常有用的设计模型。当在某个状态下有事件发生时，根据当前状态和输入事件的不同，选择如何处理该事件以及是否需要转换到下一个状态。有限状态机(FSM)是一个五元组 $M=(Q,\Sigma,\delta,q_0,Z)$，其中：

- Q 是一个状态的非空有限集合。
- Σ 表示该系统能够接收的所有事件的集合。
- δ 是 $Q\times\Sigma\to Q$ 的映射，称 δ 为状态转移函数，它描述了系统中每个状态转换到其它状态的可能性，只要该事件发生，就会发生转移，常用定义式 $\delta(q_1,a)=q_2$ 表示在 q_1 状态下接收到某个输入事件 a 之后，转入指定的新状态 q_2。
- $q_0\in Q$ 是系统的一个特殊状态，一般是系统启动时的初始状态。
- $Z\in Q$ 是终结状态的集合。

有限状态机一般有两种表示方式：状态转移表和状态转移图。通常用有向图来表示有限状态机，其节点代表状态。若在状态 q_1 接收到某个输入事件 a 后转向 q_2 状态，就在图中画一条从 q_1 到 q_2 的带箭头的弧线，并在弧线上标记 a。状态转移图如图 1 所示。

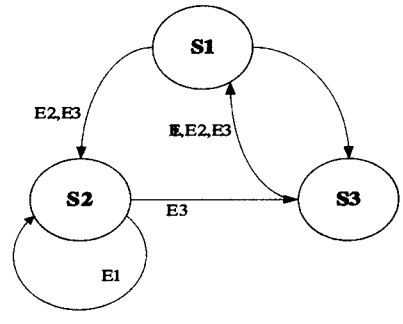

图 1

乔英东　硕士生，工程师，主要研究方向：模式识别与图像处理。肖创柏　教授，博士，主要研究方向：数字信号处理。

表 1 状态转移表

状态/事件	E1	E2	E3
S1	S3	S2	S2
S2	S2	S1	S3
S3	S1	S1	S1

3 状态机在数字电视机顶盒的应用

3.1 设计思路

数字机顶盒的各模块任务主要有两类，一种是和状态相关的任务，如用户界面模块和播放器模块，它们运行的方式一定程度上和它们现在所处的状态有关。另一种是和状态无关的任务，这些任务完全独立运行，和目前的运行的状态无关。所有的监护进程和正在运行的驱动程序就是状态无关的任务，所有的状态无关的任务在系统初始化状态时开始执行或初始化状态之前就开始运行。数字电视机顶盒软件系统是若干个担负一定任务独立进程的复合体，每个进程都有独特的功能。从概念上讲，一个任务或者进程有 3 个状态卸载状态、就绪状态和激活状态(如图 2 所示)。在开始的时候任务处于卸载状态。在状态机执行了装载操作(为程序创建一个进程)后，它转为就绪装态。在就绪状态，进程不做任何事情仅仅是等待命令。在接到一个预先定义的命令后，进程表现出一定功能。挂起和卸载操作是激活和装载相反的操作。

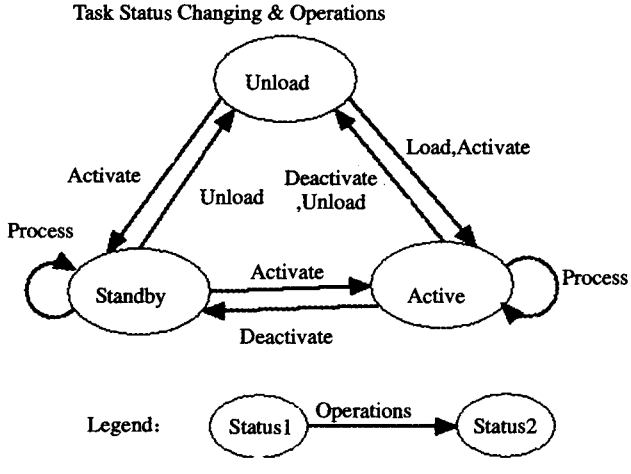

图 2

状态机的机制是状态机作为各个进程的管理者来确保机顶盒软件系统的稳定性和可靠性。一个与状态相关的进程的开始或者结束取决于状态机的要求。和通常的状态机模型一样，机顶盒的状态机确保系统保持在预先定义的几种状态下，除此之外，它还要确保，当需要时，系统能够从一种状态转向另一种状态。

3.2 状态机机制

图 3 是典型数字电视机顶盒的系统状态机的状态转移图，系统启动时进入初始化状态，用户状态是指用户界面被显示时系统的状态，播放状态是指播放器运行时的系统状态，播放用户状态是指当播放器和用户界面同时运行时系统的状态。浏览器状态是指浏览器进程处于激活状态时系统的状态。状态机一般有 4 个主要的组成部分：①状态：它定义了行为和可能产生的动作；②状态转换：一种状态向另一种状态转变；③规则或者条件：状态转换必须满足的条件；④输入事件：它既可以是内部产生的也可以是外部产生的，它能触发规则，引起状态转换。

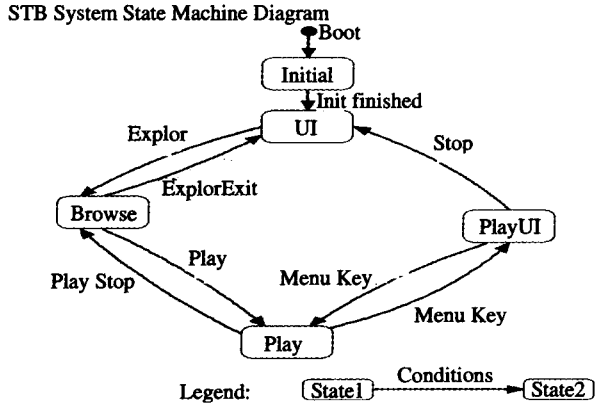

图 3

3.2.1 状态抽象

一种状态是指系统运行时的状态，状态表现为几种任务同时存在和运行。一个运行的系统可能随机地处于下面三种状态：运行在一种状态、状态的转换、状态转换后的另一种状态。一个确定的系统的状态必须是确定的预先定义的状态。根据机顶盒的特征，定义机顶盒的状态如表 2 所示。

表 2

Controls	Zero or init state	Browse state	Browseplay state	Play state
player	unload	unload	active	active
browser	unload	active	active	unload
webserver	active	active	active	active
lcdcontrol	active	active	active	active

3.2.2 状态转换

随着状态的转换，控制器状态也改变了。在机顶盒软件系统中，控制器是有一定功能的任务或者当运行时叫进程。随着任务状态的改变，系统的状态也改变。状态转换函数将比较每个控制器目前的状态和目标状态，如果两者不同，激活相应的转换行为，如果两者相同，处理操作就要被执行。

3.2.3 转换规则

实际上，当转换行为发生时，更多的条件和因素要考虑，这就使得转换变得复杂。此外，作为多任务管理程序，任务间的通信和异步问题应该被考虑。所有任务应当遵循以下准则：状态转换行为不能被中断；控制器状态的转换应该是可靠的；状态转换的结果应该是可靠的；当处于状态转换时，有关状态查询的请求应该被拒绝；资源释放的优先级应该高于资源申请的优先级。

3.2.4 输入事件

运行的程序必须顺序执行一系列的动作，根据程序运行的模型处理不同的输入。程序实现的最好方式就是状态机机制。输入事件对于状态机来讲是有意义的事件，这些事件通常会引起状态的变迁，促使状态机从一种状态切换到另一种状态。

3.3 状态机的实现

3.3.1 数据结构定义

整个状态机是由状态、事件和动作 3 种数据结构构成的。

```
typedef enum _states_s_{
    ST_ZeroState=0,
    ST_InitState,
    ST_PlayState,
    ST_UIState,
```

```
    ST_PlayUIState,
    ST_BrowseState,
    ST_BrowsePlayState,
    ST_LastState, /* Keep this Last */
} stbStateIndex_t;
```

stbStateIndex_t 定义了状态的种类，它与机顶盒真实状态间是一一对应的，是机顶盒预先定义的状态。

```
typedef enum _control_status_{
    ST_UnloadStatus=0,
    ST_StandbyStatus,
    ST_ActiveStatus
}stbCtrlStatus_t;
typedef struct {
    stbCtrlStatus_t controls [ST_NUM_CONTROLS];
} stbState_t;
```

stbState_t 定义了状态节点。状态节点是控制器的状态的集合。

```
typedef struct _MSG_s_{
    unsigned int type; //Identity of message
    unsigned int msgId; //Message value
    unsigned int session; //Unique session key for a whole request
    unsigned int pid; //Sender PID
    unsigned int size; //Size of content
    int reserved [3];
    unsigned char content
    [MSG_MAXSIZE - MSG_HEADER_SIZE];
}Msg_t;
```

Msg_t 定义了事件。这里事件是消息，状态机通过响应消息来改变自身状态。

```
typedef struct _controls_s_{
    unsigned int pid;
    stbStatus_t status;
    stbMsgFunc_t load;
    stbMsgFunc_t activate;
    stbMsgFunc_t deactivate;
    stbMsgFunc_t process;
    stbMsgFunc_t unload;
} stbControl_t;
```

stbControl_t 定义了动作，来实现状态的迁移。

3.3.2 接口函数定义

int InitSM(void)

初始化状态机。

getNextState(int curState, msg_t * pMsg, int msgHandled)

检查消息是否引起状态迁移。

void transitState(msg_t * pMsg)

目前的状态转换到下一个状态。

3.3.3 状态机的驱动

```
void MC_loop(){
    while(1){
        rcvMsg(&msg);
        NextState=getNextState(CurrentState,&msg);
        if(NextState ! =CurrentState)
        transitState(NextState);
        sleep(1);
    }
}
```

抛去一些不必要的细节，状态机通过不断接收消息来响应外界的各种事件，实现状态转移，确保系统正确运行。

结 论 嵌入式系统中，资源稀少，实时性要求高，同时随着技术的发展，应用需求越来越高，使得应用程序复杂程度也越来越高。利用有限状态机模型，可以大大简化编程，尤其是在数字电视系统中，状态通常是确定的和有限的，因此，利用有限状态机模型，可以减轻开发人员的工作量，提高软件的可读性，增强软件的可维护性和扩充性。

参 考 文 献

1 熊振云，阮俊波，金惠华. 嵌入式软件中状态机的抽象与实现. 计算机应用，23(10)

2 徐小良，等. 有限状态机的一种实现框架. 工程设计学报，10(5)

(上接第 285 页)

结论 本文在串行加法器的基础上用重写系统对不恢复余数阵列除法器进行了严格的描述和直观的证明，说明了重写归纳技术在硬件正确性验证方面存在优势。与传统的模拟、仿真相比，其明显的优点是一方面它解决了由于实际的计算机字长较长、状态较多传统方法很难穷尽模拟这一问题，另一方面它使用的数学方法证明更加严格可靠。

本文的验证方法还未实现自动化，进一步的工作包括研究使用验证工具进行自动化验证、过程中的引理推测和效率改进等。

参 考 文 献

1 韩俊刚，杜慧敏. 数字硬件的形式化验证. 北京：北京大学出版社，2001

2 Kapur D, Subramaniam M. Mechanizing Verification of Arithmetic Circuits: SRT Division. In: Proc. 17th FSTTCS, Vol. 1346 of LNCS, Springer Verlag, 1997. 103～122

3 张欢欢，邵志清，宋国新. 基于重写归纳技术的串行加法器的描述和验证[J]. 华东理工大学学报，2003，1(29)：59～64

4 张欢欢，邵志清，宋国新. 基于幂表的并行加法器的归纳验证[J]. 电子学报，2003，6(31)：932～937

5 Kapur D, Subramaniam M. Mechanically verifying a family of multiply circuits [A]. In: Proc. of 8th Conf on Computer Aided Verification [C]. Berlin: Springer-Verlag, 1996. 135～146

6 Kapur D, Subramaniam M. Using and Induction Prover for Verifying Arithmetic Circuits. J. of Software Tools for Technology Transfer. Springer-Verlag, 2000, 3(1): 32～65

7 Akbarpour B, Tahar S, Dekdouk A. Formalization of Fixed-Point Arithmetic in HOL. Formal Methods in Systems Design, Springer Verlag, 2005, 27(1-2): 173～200

8 Tahar S, Zobair M. H, Song X. Formal Verification of a SONET Data Stream Processor. IEE Proceedings - Computers and Digital Techniques, 2004, 151(1): 71～81

9 Kort S, Tahar S, Curzon P. Hierarchical Formal Verification Using a Hybrid Tool. International Journal on Software Tools for Technology Transfer, Springer Verlag, 2002, 4: 1～10

10 Baader F, Nipkow T. Term Rewriting and All That. Cambridge University Press, 1998

11 Arvind, Shen Xiaowei. Using Term Rewriting Systems to Design and Verify Processors. IEEE Micro Special Issue on Modeling and Validation of Microprocessors, 1999

12 Hoe J C, Arvind. Hardware Synthesis from Term Rewriting Systems. In: Proceedings of X IFIP International Conference on VLSI (VLSI 99), Lisbon, Portugal, November 1999

13 Mauricio Ayala-Rincon, Nogueira R B, Llanos C H, Jacobi R P. Modeling a Reconfigurable System for Computing the FFT in Place via Rewriting-Logic. In: Proceedings of the SBCCI'03. IEEE Computer Society Press, São Paulo, SP, Brazil, 2003. 205～210

14 Mauricio A-R, Nogueira R B, Llanos C H, Jacobi R P, Hartenstein R. Efficient Computation of Algebraic Operations over Dynamically Reconfigurable Systems Specified by Rewriting-Logic Environments. Conferencia Internacional de Ciencia de la Computación, Bio-Bio. 23rd SCCC/2003. 60～69

15 Mauricio A-R, et al. Modeling and Prototyping Dynamically Reconfigueable Systems of Efficient Computation of Dynamic Programming methods. In: 17th Symp. On Integrated Circuits and Systems Design. ACM Press, 2004. 248～253