

[12] 发明专利申请公开说明书

[21]申请号 94190279.X

[51]Int.Cl⁶

G06F 17/00

[43]公开日 1995年10月4日

[22]申请日 94.1.6

[30]优先权

[32]93.5.10 [33]US[31]08 / 060,771

[32]PCT / US94[33]00[31]41 94.1.6

[32]WO94 / 272[33]4 [31]英 94.11.24

[32]95.1.10[33][31]

[86]国际申请 [87]国际公布 [85]进入国家阶段日期

[71]申请人 塔里根特公司

地址 美国加利福尼亚州

[72]发明人 史蒂芬·H·米尔恩 马修·L·德曼

约翰·C·托拜尔斯

詹姆斯·M·蒂德尔

[74]专利代理机构 北京市中原信达知识产权代理公
司

代理人 余 滕 余 刚

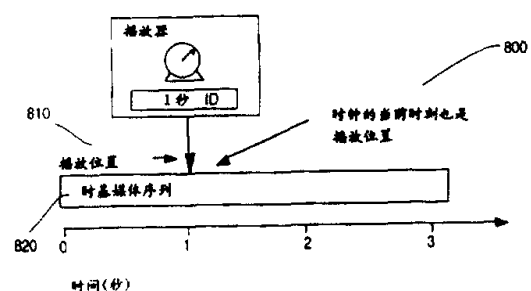
说明书页数:

附图页数:

[54]发明名称 多媒体同步系统

[57]摘要

本发明涉及对多个多媒体事件提供同步定时的系统和方法,时钟目标被限定在存储器中且与当前时刻的内部和外部源关联。时钟目标亦可在显示器上显示,但亦可在连接被定义之后隐藏起来。至少一个代表音频、视频或其他多媒体事件的多媒体目标被定义和连接到特定的时钟目标。然后,处理器把多媒体目标和相关的时钟目标或目标同步。最后,多个多媒体事件与相关的时钟同步实施。



权 利 要 求 书

1. 一个用于多媒体表示的计算机系统,其特征在于包括:

一个存储器;

一个显示器;

一个时钟目标,该时钟目标有一个位于存储器中的当前时刻且可在显示器上显示;

多个位于存储器中的多媒体目标,该多媒体目标可在显示器中显示;以及

一个用于把至少一个多媒体目标与时钟目标同步的处理器。

2. 权利要求 1 所述的系统,还包括用于通过一个图符操作启动同步的处理装置

3. 权利要求 2 所述系统,还包括通过在时钟目标上连扣两下以启动同步的处理装置。

4. 权利要求 2 所述系统,还包括通过投入而激励同步的处理装置。

5. 权利要求 1 所述系统,还包括把多个多媒体目标与时钟目标同步的装置。

6. 权利要求 1 所述系统,还包括把多个时钟目标与一个时钟目标同步的装置。

7. 权利要求 1 所述系统,还包括调整同步的颗粒性的装

置。

8. 权利要求 1 所述系统,还包括获取当前时刻的外部装置。

9. 权利要求 1 所述系统,还包括用于获得当前时刻的外部时间源。

10. 权利要求 1 所述系统,还包括迫使当前时刻反向运行的装置。

11. 权利要求 1 所述系统,其中的时钟目标可以跨越多个地址空间。

12. 权利要求 1 所述系统,还包括通过一个在显示上连接两个目标的几何图形而把时钟目标连到多媒体目标的装置。

13. 权利要求 12 所述系统,其中的几何图形是一个线段。

14. 一种在计算机上同步多媒体表示的方法,其中的计算机有一存储器和一个显示器,该方法的特征在于包括以下步骤:

(a)生成一个具有位于存储器中的相关当前时刻并可在显示器上显示的时钟目标;

(b)生成多个位于存储器中并可在显示器上显示的多个多媒体目标;

(c)把至少一个多媒体目标与时钟目标同步。

15. 权利要求 14 所述方法,还包括通过一个图符操作启动同步的步骤。

16. 权利要求 15 所述方法,还包括在时钟目标上按扣两下以启动同步的步骤。

17. 权利要求 15 所述方法,还包括通过投入而激发同步的步骤。

18. 权利要求 14 所述方法,还包括把多个多媒体目标与时钟目标同步的步骤。

19. 权利要求 14 所述方法,还包括把多个时钟与一个时钟同步的步骤。

20. 权利要求 14 所述方法,还包括调整同步的颗粒性的步骤。

21. 权利要求 14 所述方法,还包括从一个外部设备获得当前时刻的步骤。

22. 权利要求 14 所述方法,还包括询问获得当前时刻的多个外部时间源的步骤。

说明书

多媒体同步系统

本发明涉及计算机系统的改进,尤其涉及对各种多媒体事件的定时进行同步处理的系统。

本申请与欧顿等人(Debra L. Orton David B. Goldsmith, Christopher P. Moeller 以及 Andrew G. Heninger)于 1992 年 12 月 23 日提交到美国联邦专利商标局并转让给本申请人(Taligent, Inc.)的题为“面向目标的结构系统”有关,该专利申请已于 1993 年 12 月 23 日提交到中国专利局,其申请号为 93112880.3 该专利申请的内容为本申请所引用。本申请的部分内容受版权保护。版权所有人允许在专利商标局中该专利申请作为专利文件被传真复印,但保留所有其它的版权。

多媒体也许是发展最快的计算机系统应用。越来越多的用户采用计算机向终端用户表示图形,声音以及图象信息。用户越来越希望有用于管理多媒体表示的人类工程学界面。过去,系统时钟通常被用于在一特定时刻开始播放声音。但是,使声音或音乐表达与信息显示同步的工具作为未折合(unfolded)的多媒体表达是不可能的。

现有的不具备本发明的同步能力的多媒体系统的例子包括苹果公司的“Quicktime”以及微软公司的 Video for Win-

dows”(刊载于“NEWMEDIA”1993 年 3 月期第 36 页到 42 页),题为“It's Showtime”。IEEE Spectrum 在 1993 年 3 月期第 22 页到 31 页以“Interactive Multimedia”和 32 页到 39 页以“The Technology Framework”为题讨论了解决先有技术中的同步问题的重要性。这两篇文章指出了多媒体内部操作性的重要性。而该内部操作性则是由本发明的同步性能表达的。

因此,本发明的主要目的是提供一种利用具有存储器和显示器的计算机在多媒体表达的全过程对各种多媒体事件进行同步的系统和方法。在存储器中定义与当前时刻的内部或外部源关联的时钟目标,该时钟目标可在显示器上显示,但是,一旦其连接被定义,它们也能被隐藏。代表音频,视频或其他多媒体事件的至少一个多媒体目标被定义并连接到一个或多个特定的时钟目标。然后,处理器使这些多媒体目标与相关的时钟目标同步。

图 1 是根据本发明最佳实施例的个人计算机框图;

图 2 展示了根据本发明的一个最佳实施例的时钟目标;

图 3 是根据本发明的一个最佳实施例的系统定时器;

图 4 示出了根据本发明的一个最佳实施例的时钟目标的定时输入的外部源;

图 5 示出了根据本发明的一个最佳实施例的主时钟目标和从时钟目标;

图 6 示出了根据本发明的一个最佳实施例用于驱动时钟目标层级的时钟源;

图 7 示出了根据本发明的一个最佳实施例的时基序列;

图 8 示出了根据本发明的一个最佳实施例的时基媒体播放器；

图 9 示出了根据本发明的一个最佳实施例的多媒体播放器,该播放器与主时钟外部同步。

图 10 是根据本发明的一个最佳实施例作为另一个时钟目标的主时钟目标,该另一时钟依次成为再一个时钟的主时钟；

图 11 示出了根据本发明的一个最佳实施例的音频和视频序列同步；

图 12 示出了根据本发明的一个最佳实施例的音频播放器(主播放器),该播放器与一视频播放器(从播放器)同步；

图 13 示出了根据本发明的一个最佳实施例的音频序列,其中的视频序列被同步以便以比音频序列快一倍的速率播放。

图 14 示出了根据本发明的一个最佳实施例的计算机显示,其中,一个视频序列被同步以便以比音频序列快一倍的速率播放。

图 15 示出了根据本发明的一个最佳实施例的主从播放器目标；

图 16 示出了根据本发明的一个最佳实施例的调节旋钮；

图 17 示出了根据本发明的一个最佳实施例的调节旋钮,该旋钮作为多个多媒体目标的控制器；

图 17B 和 17C 是根据本发明的一个实施例的调节旋钮

的流程图；

图 18 展示了根据本发明的一个实施例出现在计算机显示器上的时钟层级；

图 19 示出了根据本发明的一个实施例通过定时源而间接与另一时钟相连的时钟；

图 20 是根据本发明的一个实施例的 SetTime 逻辑的详细流程图；

图 21 是根据本发明的一个实施例的 TickleWakeups 功能的逻辑流程图；

图 22 是根据本发明的一个实施例的 Get Current Time 操作的逻辑流程图；

图 23 是根据本发明的一个实施例的延迟和告警处理的逻辑流程图；

图 24 是根据本发明的一个实施例 Sync To Keep My Time 处理的逻辑流程图；

图 25 示出了根据本发明的一个实施例的音频播放器；

图 26 示出了根据本发明一个实施例作为控制器的音频播放器；

图 27 是根据本发明的一个实施例受控于一个时钟的音频播放器的流程图；

图 28 示出了根据本发明的一个实施例的日出图象序列；

图 29 是根据本发明的一个实施例与一图形播放器有关的逻辑；

图 30 示出了根据本发明的一个实施例的 MIDI 序列和 MIDI Track 格式;

图 31 示出了根据本发明的一个实施例通过一个驱动器播放一个序列的 MIDI 播放器;

图 32 是根据本发明的一个实施例的 MIDI 播放器的详细逻辑流程图;

图 33 示出了根据本发明的一个实施例的空白桌面显示;

图 34 示出了根据本发明的一个实施例的选定时钟目标定义显示;

图 35 示出了根据本发明的一个实施例的各个时钟目标和多媒体目标;

图 36 示出了根据本发明的一个实施例的各个连接到一起的时钟目标和多媒体目标;以及

图 37 示出了根据本发明的一个实施例的可见目标,该可见目标与一个音频目标同步。

本发明最好在 IBM PS/2 或 Apple Macintosh 计算机之类的个人机中的操作系统里实施。图 1 示出了具有代表性的硬件环境,它展示了根据本发明的工作站的典型硬件构造,它包括一个中央处理器 10(例如常规的中央处理器),以及多个与系统总线 12 相连的其他部件。图 1 所示工作站包括一个 RAM14,ROM16,用于连接磁盘部件 20 之类的外围设备到总线的 I/O 适配器 18,用于把键盘 24,鼠标器 26,扬声器 28,麦克风 32 以及/或其他接口设备(例如触摸式屏幕

设备)等连到总线的用户接口适配器 22,用于把工作站连到数据处理网络 23 的通信适配器 34,以及把总线连到显示设备 38 的显示适配器 36。该工作站中的操作系统为 Apple System/7 一类的操作系统。

在一个最佳实施例中,本发明是以采用面向目标的编程技术的 C++ 程序语言实现的。正如本领域一般技术人员所理解的,面向目标的编程(OOP)目标是包括数据结构和数据中的操作的软件实体。这些单元一起以数据单元表示的特性以及由其数据操纵功能表示的行为使目标实际成型为任意现实社会中的实体。以这种方式,目标可以成型为具体的事物,例如人和计算机,而且亦可成型为抽象概念,例如数字或几何概念。目标技术的好处来自于三个基本原理:封装,多形性以及继承,所谓封装(encapsulation)是指一种把数据以相应的过程存储起来的技术,而多形性则是指在一个共用的界面后面隐藏不同实现。从而简化目标之间的通信的能力。继承则是指一种处理,其中,分类可以利用在分类层级的分支上高于该分类的所有分类所限定的方法和变量。

目标隐藏或封装其数据的内部结构和算法。除了展现这些实施细节,目标展示清楚表达其抽象而无冗余信息的界面。多形则将封装又推进一步。该概念是多个形状,一个界面。一个软件部件(Component)可以提出另一部件的请求而无需知道那一部件倒例底是怎样的,接收请求的部件对其进行解释可根据其变量和数据确定如何执行该请求。第三个原理是继承,它允许开发者再使用先有的设计和编码。这一能

力使开发者避免擦除以生成软件。进而,通过继承,开发者获得继承了行为的子分类,从而使开发者将其客户化以满足特定的需求。

一种先有技术的方法是在过程环境中对目标和分类库进行分层,市场上的许多应用结构都采用了这种设计方法。在这种设计中,在单一操作系统的顶部有至少一个层级。虽然这一方式利用了目标层级中的封装,多形性以及继承,而且是对过程编程技术的重大改进,但其仍有限制,其困难之处在于虽然开发者很易再使用他们的目标,但其他系统则很难使用这些目标而且开发者仍需由过程操作系统(OS)调用进入较低的非目标层级。

面向目标编程的另一方面是应用开发的结构方式,结构的最合理定义之一是由依利诺易斯大学(University of Lllinois)的强森(Ralph E. Johnson)以及普渡(Purdue)的卢梭(Vincent F. Russo)做出的在他们1991年发表的题为“面向目标的设计的再使用(Reusing Object—Oriented Designs)”(该文发表在依利诺易斯大学的技术报告 UIUCDCS91—1696 页)一文中提出了以下定义:“一个抽象分类是一组合作执行一组响应的目标的设计。因此,一个结构是一组合作执行定义组的计算响应的目标分类。从编程观点出发,结构实际上是一组提供工作应用程序的预制结构的内连目标分类组。例如,一个用户界面结构可以提供绘制视窗,卷动条以及选项屏等的支持和“系统预置”行为。由于结构基于目标技术,该行为可被继承和置换以使开发者对结构进行扩展并在一特

定的区域生成客户化方案,由于编程不改变初始码而只是扩展软件,上述优点便成为一个优于传统编程的主要特点。此外,由于结构提供了体系结构指导和模型并在同时允许为问题范畴提供了唯一的独特行为,开发者不会盲目地通过编码层级进行工作。

从商业观点来看,结构被视为在一特定知识领域封装或嵌入专门知识的方式,公司开发机构,独立软件销售商(ISV)以及系统集成者已经提出了在特定区域中的专门知识的要求,这些特定区域包括制造,财会,以及现金交易。该专门知识被嵌入编码中。结构允许这些开发机构捕捉并封装该专门知识的通用特性,其过程是将其嵌入机构的编码之中,首先,这使得开发者生成或扩展利用该专门知识的应用程序。因此把问题解决一次并使商业规律和设计被前后一致地实施和使用。另外,在结构后面的嵌入专门知识以及结构对已经在统管生产和销售的市场的知识(例如生产,财会或生物技术)的那些机构有策略的有益含义,或生物技术需要包装,再销售,以及推广应用其知识,并进而推广和开发普及其技术。

从历史而言,结构(framework)只是在最近才成为个人计算机的主流概念。这一发展得力于面向目标语言(例如C++)的发展。C++最早是在UNIX系统和研究人员的工作站上实现的,而不是在商用个人计算机上发现的。正是C++语言以及其他面向目标的语言(例如Smalltalk等等)使得许多大学和研究项目产生了当今商用结构和分类库的先驱。这些例子包括斯坦福大学(Standford University)的 InterViews,卡

内基—麦伦大学 (Carnegie — Mellon University) 的 Andrew toolkit, 以及 苏黎世大学 (University of Zurich) 的 ET++ 结构。

根据想解决的问题以及涉及的系统层级, 有多种结构。这些结构包括帮助开发用户界面的应用程序结构以及提供基本系统软件服务 (例如通信, 打印, 文件系统支持, 图形等等) 的低级结构, 应用结构的商用例子是 MacApp (Apple), Bedrock (Symantec), OWL (Borland)、NextStep App Kit (NeXT), 以及 Smalltalk—80 MVC (ParcPlace) 等等。

结构编程要求与常规的其他系统不同的新思维方式。事实上, 它与传统概念的“编程”全然不同。在以前的操作系统中 (例如 DOS 或 UNIX) 开发者自己的程序提供了所有结构, 操作系统通过系统调用提供服务——开发者的程序在需要服务时以及服务完成控制返回时进行调用。程序结构基于嵌入开发者写的编码中的控制流程。

当结构使用时, 其过程相反, 开发者不再依赖控制流程。开发者必须根据执行流程事先了解编程任务的趋向。进而, 必须依据目标的响应性进行思考, 这必须根据结构确定何时执行任务。由开发者所写的程序由不是开发者写的而且开发者从未见到的编码激励。控制流程中的转换可能对只在过程编程中有经验的开发者构成较大的心理障碍。但是, 一旦明白之后, 结构编程所需工作就大大少于其他编程。

以一个应用结构 (application framework) 给用户 提供预制功能性的同样方法, 系统结构 (正如在最佳实施中包括的)

通过提供系统级服务而实现同样概念。系统程序员之类的开发者可用其进行子分类/置换以生成客户化的方案。例如,考虑一个可以提供基础以支持视频,音频,MIDI 动画等等新的以及多功能设备的多媒体结构需要支持一种新设备的开发者必须写一个设备驱动程序。若以结构完成这一任务,则开发者只需支持该新设备的特定特性和行为。

在这种情况下,开发者提供将为该多媒体结构调用的某种成员功能实施。开发者可得到的好处在于每类设备需要的通用编码已由多媒体结构提供了。这意味着开发者要编写,测试和检测的设备驱动程序编码减少了。使用系统结构(System framework)的另一例子是对于计算机系统接口(SCSI)设备,NuBus 卡,以及图形设备提供各自的 I/O 结构。由于有继承功能性,每个结构为在其设备类别中发现的通用功能性提供支持,其他开发者则可依据对各种设备一致的界面。

本发明的一个最佳实施例采用了结构的概念并将其应用于整个系统。对于商业和企业开发者,系统集成商或 OEM,这意味着象 MacAfpp 一类的结构所展示的优点可以不仅作用于文本和用户界面之类的事物,而且亦可用于系统的服务,例如图形,多媒体,文件系统,I/O,以及测试等等。

在一个最佳实施例的体系结构中的应用程序生成基本与编写附加到结构协议的特定域的谜块类似。在这一方式中,编程的整个概念改变了,代之以在调用多个 API 层级的编码行之后写编码行,软件将由在该环境中的先有结构导出的分类所发展,然后根据需要加入新行为并/或替换继承的

行为。

如此,开发者的应用程序成为由所有其他结构应用程序编写和共享的编码集。由于开发者将可在相互的工作上工作,因此这是一个很有力的概念,这也为开发者提供了根据需要进行客户化的灵活性。某些结构将被原样使用。在某些情况下,客户化的量将很小,所以开发者插入的谜块将很小。在另外的情况中,用户可以进行很大修改并生成全新的事件。

在一个最佳实施例中,如图 1 所示,软件时钟负责提供时基,而位于 RAM14 中并由 CPU10 控制(或通过总线 12 或通信适配器 34 外接)的多媒体播放器则负责跟踪时钟。无需中央播放器来协调或管理系统的整个处理。这一体系结构在新的媒体类型加入时很灵活并可扩展。由本发明的最佳实施例实现的系统和方法可提供以下特性。

- 播放器可经常进行检查以确保应用程序要求的准确同步级。
- 多个时钟可相互同步,而且时钟间的线性时间关系可用于指示各个序列相互间的速度及偏移。
- 多个时间源。任何时钟(以及任何播放器)可以不仅与系统时钟同步,亦可与外部时间源(例如来自录象带的时间码)同步,或与用户动作(例如调谐软件结构的微调/往复旋钮等其他控制)同步。
- 时钟可以向后倒退。
- 时钟可跨越多个地址空间。

术语定义

何谓时钟?

如图 2 所示,软件时钟是一个执行以下功能的目标。

- 时钟有“当前时刻”,由时间目标 200 所示,时间目标是一个浮点数,它以微微秒(10^{-12})的精度进行时间测量。为测定该当前时刻而提供了成员功能。一个成员功能是作用于一个目标的功能。当前时刻在时钟正转时可以增加。但其值亦可在时钟反转时减小。

- 延迟:时钟可以在某一时刻停止一段时间,该时间叫做“延迟时刻”。若时钟正转,则当时钟的当前时刻大于或等于延迟时刻时,但时间不会停止;若时钟反转,则在当前时刻小于或等于该延迟时刻时,时间不会停止。

告警:时钟可以在一特定时刻把一个 IPC(内部处理通信)消息送到一个端口,该时刻称为告警时刻。若时钟正转,则在时钟的当前时刻大于或等于该告警时刻时送出该 PIC 消息。若时钟反转,则在时钟的当前时刻小于或等于该告警时刻时送出该 IPC 消息。

- 多个时间源:通常,时钟的时间先于基于系统定时器的实时。该系统定时器由图 3 的时间源目标 300 表示,但是,亦可将时钟同步到另一时间源,例如从一个录象设备(VTR)进入计算机的电影和电视工程师协会(Society of Motion Picture and Television Engineers (SMPTE))时间码,如图 4 所示。时钟的当前时刻将跟踪来自 VTR 的时间码,当 VTR 加速时,时钟也会以时钟速率加速。而当 VTR 减速时,时钟亦会

降低其速率。当 VTR 反转时,时钟亦会反转。时钟会总是与 VTR 保持同步。

一个时间源亦象时钟一样有当前时刻,一个线性函数可用来指示时间源当前时刻和时钟的当前时刻之间的关系。

$$T_{\text{clock}} = a \cdot T_{\text{timesource}} + b;$$

这里, T_{dock} 是时钟的当前时刻;

$T_{\text{timesource}}$ 是时间源的当前时刻;

a 是确定时钟当前时刻相对于时间源的当前时刻的比率的浮点值;

b 则是确定时钟当前时刻相对于时间源的当前时刻的偏移的时间目标。

该函数称为“时钟函数”。

• 同步:时钟亦可相互同步,给定两个时钟,一个主时钟和一个从时钟,则时钟函数可由一个时间源表示;

$$T_{\text{slave}} = a \cdot T_{\text{master}} + b;$$

T_{slave} 是从时钟的当前时刻;

a 是确定从时钟当前时刻相对于主时钟当前时刻的比值的浮点值;

b 则是确定从时钟当前时刻相对于主时钟当前时刻的偏移的时间目标。

例如在图 5 中,时钟 510(从时钟)与主时钟 500 同步。从时钟可以是另一时钟的主时钟,一个时钟可以是任意数量的从时钟的主时钟。由于这两条定理,可以生成时钟的随机树结构。图 6 示出了用于驱动软件时钟层级的时间源。

提供了两个时钟成员功能以便将时钟同步。它们在从时钟上调用。一个成员功能叫做 SyncTo(), 它使客户得以指定主时钟, 比率以及偏移。第二个则称为 SyncToKeepMyTime(), 它使客户得以指定主时钟和比率, 但计算偏移从而使从时钟的当前时刻在同步之时不会改变。

此外, 成员功能 SetRate 可用于直接设定从时钟相对于其主时钟的比率。

在不同地址空间的时钟目标和时间源目标可被同步到一起。

- 起动和停止, 时钟可被停止, 在这种情况下其当前时刻不变, 而无论其主时钟是否改变, 一个停止的时钟可被再次启动, 从而致使该时钟继续跟随其主时钟。其比率未改变。它保持为时钟停止前的比率。但在时钟启动时偏移改变了, 它的当前时刻即为时钟停止时的时刻。

- 设定速率: 时钟的功能之比可设定为任意浮点值。当比率改变时, 偏移也改变但时钟的当前时刻不改变。

何谓时基媒体序列

一个时基媒体序列是一个抽象基本分类, 该分类用于代表一部分音频, 视频, 动画或乐器数字接口 (MIDI) 数据, 或任何随时间变化的其他数据。其始于时刻 0 并有一由一时间目标代表的期间, 图 7 即为 3 秒期间的时基序列。时基媒体序列的子分类用于实施音频, 视频和 MIDI 序列。

何谓时基媒体播放器

一个时基媒体播放器(此后称之为播放器)是一个抽象

基本分类;它用于播放和记录一个时基媒体序列。这一分类可以是子分类以生成音频,视频和 MIDI 的播放器,图 8 示了一个时基媒体播放器的例子,一个播放器有一个相关的软件时钟目标。软件时钟的当前时刻 800 代表了播放器的播放位置 810,播放位置 810 与一个磁带记录器上的磁头相类似。通常,播放器的时钟与一个系统预置时钟或系统定时器之类的时间源同步。这样一个播放器被认为是“内部被同步的”。但是,播放器的时钟亦可与另一时间源或时钟(包括另一播放器的时钟)相同步。这样的播放器被认为是“外部同步的”。

图 9 中,播放器 900 被外部同步于主时钟 910。主时钟 910 根据其与从时钟的同步而确定播放器 900 的播放位置。

例如:播放器的时钟与主时钟同步,从而:

$$T_{\text{player}} = 1 \cdot T_{\text{master}} + O$$

若主时钟变慢,播放器的播放速度以时钟步进降低;若主时钟加快,则播放器加速。在所有情况下播放器与主时钟保持同步。

当外部同步时,播放器负责确保其播放位置总是反映播放期间的时钟当前时刻。这一情形也同样发生在记录期间。

播放器的时钟亦可用作作为一个对于外部时钟的主时钟,该外部时钟依次可以作为另一时钟的主时钟。图 10 中,播放器 A 的时钟 1000 作为时钟 X1010 的主时钟。时钟 X1010 又作为播放器 B 的时钟 1020 的主时钟。当播放器 A1000 播放时,时钟 X1010 和播放器 B1020 跟随播放器 A 的时钟。当播

放器 A 停止时,其时钟不动且时钟 X 或播放器 B 的时钟亦不动。当外部同步时,播放器的责任是确保其时钟的当前位置总是反映播放位置。

同步时基媒体序列

主/从播放器

为了把音频和视频序列同步到一起,如图 11 所示,两个播放器的时钟应按图 12 所示进行同步。视频播放器将总是跟随音频播放器。若音频播放器加速,则视频播放器也加速,若音频播放器减速,则视频播放器亦减速。若音频停止,则视频亦将停止。若音频开始反转,则视频亦反转。任意两个时基媒体播放器均可以此方式同步,而不仅限于音频和视频。

播放器和共用时钟同步

为了在一个音频序列之后一秒启动一个视频序列(其中该视频序列以比音频序列快一倍的速度播放),如图 3 所示,则需生成两个播放器且其时钟必须与一个主时钟同步,如图 14 所示。任意多个时基媒体序列可与一个主时钟同步。

菊花链播放器

一个播放器可以既为控制源亦为受控源,如图 15 所示,这类播放器是外部同步的,而且因此必须象一个受控源一样。同时可在没有外部影响情况下作为一个控制源。一个受控源 1520 只可与主/从播放器 1510 的时钟相连。多个播放器可以被菊花链按在一起。

实施软件微调/往复旋钮速率控制器

一个微调/往复旋钮是一个可用于控制时基媒体序列集

的播放速率的图形装置。当其处于“往复”模式时,旋钮的位置控制了播放速率。当其处于微调模式时,该旋钮控制播放位置。可用软件时钟实现一个软件微调/往复旋钮。可将该微调/往复施钮的图形表示绘制于计算机屏幕之上,如图 16 所示,并由一个鼠标器所控制,当处于微调模式时,屏幕上的控制器旋钮 1610 的位置可被译为序列中的时间位置,一个时间源被用于表达该时间位置。每次当该旋钮设定于新位置时,时间源的当前时刻被设定于该序列中的相应时间位置。

一个时钟目标受控于该时间源。然后,通过把至少一个播放器与时钟目标同步,它们将跟随图 17 所示的旋钮所确定的位置。当处于往复模式时,时钟目标由系统时间源控制,屏幕上的旋钮的位置可被译为是播放速率。每次当用户改变屏幕上的旋钮位置时,相应速率就设定在时钟上。设定该时钟速率则致使所有同步的播放器相应地改变它们的速率。

图 17B 是根据一个最佳实施例与往复控制操作有关的详细逻辑流程图。当鼠标器按钮在功能框 1700 处被测到是已按下时,在判别框 1702 处进行测试以决定是否鼠标器光标位于图 16 中所示往复控制器的旋钮 11610 上。若是,则在功能框 1704 处将往复时钟速率设为零并把往复时钟设定为等于零当前时刻。然后,在功能框 1706 处,播放器受控于往复时钟并在功能框 1708 处启动播放器,最后处理完成于 1710 处。

当鼠标器位置的改变在功能框 1720 处被测到时,则在功能框 1722 处计算在中午 12 点和旋钮位置之间的角度,并

在判别框 1724 处进行测试以确定是否该角度为负值。若是, 则在功能框 1728 处把符号设为 -1 , 角度 $d = -d$, 然后控制去往判别框 1730。若测试在功能框 1726 处失败, 则符号被设定为 1 , 然后在判别框 1730 处进行测试以确定是否 $d \geq 0^\circ$ 但 $< 80^\circ$, 然后速率被设定等于符号 $\times d/80^\circ$ 且控制去往功能框 1742 以便将往复时钟速率设定为速率。若否, 则在判别框 1736 处进行测试以确定是否 $80^\circ \leq \alpha < 100^\circ$ 。然后速率被设定为等于符号 $\times 1.0$ 且控制去往功能框 1742 以便将往复时钟速率设为速率。若否, 则速率设定为等于 $\times 1 + (d - 100^\circ)/80^\circ$ 且控制去往功能框 1742 以设定该往复时钟速率等于速率。

当鼠标被释放时, 如功能框 1744 所示, 则播放器在功能框 1746 处停止且该播放器受控于播放器的内部时钟, 如功能框 1748 所示。

图 17C 是根据一个最佳实施例的微调模式操作的详细逻辑流程图。若在功能框 1750 处测得鼠标器按钮被扣下, 则在功能框 1752 处进行测试以确定是否鼠标器光标在旋钮上。若不在, 则处理终止于 1762。若在, 则在功能框 1754 处把微调时间源当前时刻设定为零, 最后的旋钮位置设定等于中午; 每度时间在功能框 1756 处设定等于 $2 \text{ 秒}/360^\circ$, 在功能框 1758 处使播放器受控于微调时间源并在功能框 1760 处启动该播放器。

若鼠标器按钮在功能框 1764 处改变了位置, 则在功能框 1760 处计算当前旋钮位置。在最后旋钮位置和当前旋钮

位置之间的偏差则在功能框 1768 处计算。而时间差在功能框 1770 处计算,且微调时间源当前进刻在功能框 1772 处被设定等于时间差。

当鼠标器按钮在功能框 1774 处释放时,便在功能框 1776 处停止播放器,而且播放器在功能框 1780 处被设定受控于其内部时钟。

时间源和时钟目标的实现

时间源实现

一个时间源和所有与其同步的时钟组叫作时钟层级。图 18 是一个最佳实施例的计算机显示上的时钟层级展示。时间源必须能提供其当前时刻和实现告警和延时,告警和延时叫做“唤醒”。送出一个 IPC 消息(一个告警)或中止一段时间(延时)叫做启动一个唤醒。时间源负责启动设置在时钟层级中任何时钟上的任意的和全部的唤醒。

对于时钟层级上的每个时钟,时间源都保持一个“直接函数”,该函数用于把时间源的当前时刻转换成时钟的当前时刻。直接函数具有以下形式:

$$T_{\text{clock}} = a * T_{\text{timesource}} + b;$$

这里: T_{clock} 是时钟的当前时刻

$T_{\text{timesource}}$ 是时间源的当前时刻。

a 是确定时钟的当前时刻相对于时间源的当前时刻的速率的浮点值。

b 是确定时钟的当前时刻相对于时间源的当前时刻的偏移的时间目标。

“反向直接函数”用于反向转换,即从时钟的当前时刻到时间源的当前时刻的转换:

$$T_{\text{timesource}} = (T_{\text{clock}} - b) / a$$

当一个时钟通过至少一个其他时钟间接连到一个时间源时,可由几何置换来计算直接函数。例如,考虑图 9 中的时钟 Y,它是与具有以下等式的时间源 1920 相关的一个函数:

$$T_{\text{clockx}} = a_x * T_{\text{timesource}} + b_x$$

时钟 Y1900 相对于时钟 X1910 的函数为: $T_{\text{clocky}} = a_y * T_{\text{clockx}} + b_y$ 时钟 Y1900 的直接函数可由置换所计算:

$$\begin{aligned} T_{\text{clocky}} &= a_y * (a_x * T_{\text{timesource}} + b_x) + b_y \\ &= (a_y * a_x) * T_{\text{timesource}} + (a_y * b_x + b_y) \\ &= a * T_{\text{timesource}} + b \end{aligned}$$

这里, $a = a_y * a_x$

$b = b_y * b_x + b_y$

设置时间源的当前时刻

有两种时间源,即受驱动的和不受驱动的受驱动时间源由客户设置的当前时刻而持续更新。通过调用驱动时间源的 SetTime() 成员功能可完成这一任务。图 20 示出了实现 SetTime() 的流程图。处理始于 2000,在该处,“新时间”被送至 SetTime() 并在功能框 2010 处保存该新当前时刻。

通过在判别框 2020 处比较“新时间”和时间源的以前的当前时刻,可以确定时间源的方向。若其为大,则时间源正向并且控制去往判别框 2040,若其较小,则时间源反向且控制去往判别框 2030,若其相等,则时间源的方向保持不变。“下

一时刻”是需要启动的下一唤醒。若时间源正向而且“下一时刻”小于或等于当前时刻,则稍后会有唤醒。若时间源是向反且“下一时刻”大于等于当前时刻,则稍后会有唤醒。标记有“反馈唤醒”的功能框 2050 启动以后的所有告警和延时。有关启动唤醒的细节将在下面的“时间源的唤醒表”部分予以详述。处理终止于框 2060。

一个非驱动时间源无需被保持告知何为当前时刻,系统定时器等可用于时间源。在系统定时器中,基本操作系统保持跟踪当前时刻。非驱动时间源知道如何发现其当前时刻。而且它有一个成员功能,GetNextTime(),它返回应被启动的告警或延时的下一时刻。只要在时间源中加入一个新的告警或延时,就调用 GetNextTime()。当要启动一个新的告警或延时之时,可以调用时间源的反馈唤醒()成员功能从而启动以后的任何唤醒。这便使实施者得以生成无需驱动时间源要求的恒定 SetTime()调用的一个时间源。

获得时间源的当前时刻

时间源可以很快地获得其当前时刻。其原因在于驱动时间源有一个存在一个内部数据结构中的值。而对于非驱动时间源而言,它可以查询基本操作系统。

处理唤醒

延时和告警唤醒都由时间源以类似方式处理。时间源在某一时刻(称之为唤醒时刻),从将被启动的唤醒的时钟得到一个请求。该唤醒时刻由一个反向直接函数从请求时钟的时基被转换成时间源的当地时基。时间源检测是否以后有唤醒

而且是否需要被启动。若以后有唤醒,时间源将启动它。否则,时间源把唤醒加入到唤醒的时序表(叫做唤醒表)中。

时间源的唤醒表

只要当唤醒表由于去除或加入了一个唤醒而改变时,就调用时间源的 `GetNextTime` 方法并计算下一次启动一个唤醒的时间。若该时间源是驱动时间源,则其成员功能 `SetTime()` 在有至少一个其后的唤醒时就调用反馈唤醒(`TickleWakeups()`)。 `TickleWakeups()` 检查整个唤醒表并启动所有以后的唤醒。当一个唤醒被启动时,它便从该表中被去除。一个非驱动时间源的 `GetNextTime()` 成员功能可被置换以发现何时需要调用 `TickleWakeups()` 并然后把一个定时器设为在那时候消失。当定时器消失时,时间源调用 `TickleWakeups()`”

图 21 展示了 `TickleWakeups()` 功能的详细逻辑。处理始于 2102 并立即去往功能框 2104,在该处改变变量被设定等于“伪”。然后,在功能框 2120 处获得第一个唤醒并在功能框 2160 处进行测试以确定是否唤醒到时,若是,则在功能框 2150 处启动唤醒,并在功能框 2140 处获得下一唤醒。改变的变量被设定为“真”,且控制返回判别框 2160 等待下一唤醒。

若唤醒在判别框 2160 处来到时,则在判别框 2170 处测试改变的变量。若变量改变为“真”,则在功能框 2180 处得到下一时刻。若“伪”,则控制去往 2190 结束处理。

软件时钟实施

获取一个软件时钟的当前时刻

为获取时钟的当前时刻,时钟过程首先从时钟层级的时钟源里获得当前时刻,然后获取直接函数并将时间源的当前时刻转换成时钟的当前时刻,如图 22 的逻辑流程图所示那样。该处理始于 2200 并立即去往功能框 2210 以获取时钟层级的时间源,然后在功能框 2220 处获取时间源的时间并在功能框 2230 处获取直接函数。最后,在功能框 2240 处把直接函数用于时间源的时间,从而导致在 2250 处返回的当前时刻。

由于时钟和时间源可在不同的地址空间,所以时间源的当前时刻和时钟的直接函数都存在两个地址空间之间所共用的存储器中。采用一个信号量(semaphore)来确保直接函数和时间源的时间在从共享存储器中读出时没有改变。

在软件时钟上设定延时

有两种延时,一种叫做“DelayUntil()”,它是时钟的成员功能。该延时将阻止客户的任务直到时钟的当前时刻到达一特定值。第 2 种延时是叫做 Delay For()的时钟的成员功能。它将客户任务阻止直到某一特定的时间量过去。第二种延时实际上等于调用第一种延时,它指定了一个等于时钟当前时刻加上需要的消逝时间量。

为了延迟时钟,时钟送出想要延迟的时间直到时钟层级的时间源。时间源采用时钟的反向直接函数将其译为时间源的当地时间(并且如果它是一个 Delay For()而且不是 DelayUntil()的话,加入当前时刻)。该请求被加入唤醒表(正如已在“处理唤醒”部分讨论过的),而且调用程序被停止直到

唤醒被启动。

在软件时钟上设定一个告警

在时钟上设定告警与在时钟上设定延时完全一样,不同之处只在于调用程序不被停止,代之以消息被送到一个特定的端口。图 23 是一个展示如何实现延时和告警的流程图。处理始于 2300,在该处输入输入时间和接收者。然后在功能框 2310 处获取直接函数并在功能框 2320 处应用直接函数的反向,然后在功能框 2340 处进行测试以确定是否该延时为 Delay For。若是,则在功能框 2342 处获得当前时刻。该当前时刻被加到时间中,且控制去往功能框 2346。

若不是 DelayFor,则在功能框 2346 处生成具有时间和接收者的唤醒。然后在功能框 2350 处进行另一测试以确定是否唤醒到时。若否,则该唤醒在功能框 2352 处被加到唤醒表,并在 2372 处执行 GetNextTime。然后控制回到功能框 2370 处进行删除。处理终止于 2392。若唤醒在处理框 2350 处已到时且若唤醒是一延时,则在功能框 2354 处对任务解锁并在功能框 2360 处去除唤醒。最后处理结束于 2392。若唤醒是一告警,则在功能框 2356 处送出一消息并在功能框 2360 处去除该唤醒。处理最后结束了 2392。

时钟的同步

时钟同步在软件时钟和时间源中实施,客户用两个软件时钟成员功能进行时钟同步。它们都在从时钟上调用以便将其与一个主时钟同步。一个叫做 SyncTo()的成员功能使用户指定主时钟或时间源,速率,以及偏移。二个成员功能叫

做 SyncToKeepMy Time(),它允许客户指定主时钟以及速率,但计算偏移从而使从时钟的当前时刻在同步的时刻不改变其值。

SyncToKeep My Time()

SyncToKeepMyTime()允许客户指定主时钟和一个速率,但计算偏移以致于在同步时刻从时钟的当前时刻之值不会改变。图 24 展示了该功能的详细逻辑,处理始于 2400 并立即去往判别框 2420 以确定是在主时钟位于同样的时钟层级。若否,则在功能框 2410 处把时间源之间的所有后续同步延迟并在判别框 2430 处进行测试以确定是否该主时钟受控于另一从时钟,若是,则通过功能框 2454 返回一个错误且控制去往终端 2499。

若否,则在功能框 2432 处获得从时钟的当前时刻。所有有关时钟的信息在功能框 2434 处被抽取出来,在功能框 2436 处更新新的主时间源,在功能框 2438 处获取主时钟的直接函数,该函数根据当前时刻和提供的速率在功能框 2440 处计算从时钟的函数。直接函数根据主时钟的直接函数在功能框 2442 处计算,时钟唤醒在 2444 处再次被计算,然后在判别框 2446 处进行检测以确定是否有方向变化。若是,则所有唤醒在功能框 2448 处被启动。若否,则处理去往 2450 以重复每个从时钟的处理并在功能框 2452 处提供时间源之间的同步。处理结束于 2499 处。

若主时钟在判别框 2420 中确定位于相同时间源中,则在判别框 2460 中进行另一检测以确定是否主时钟现在受控

于另一从时钟。若否,则在功能框 2462 处获取从时钟的当前时刻,在功能框 2464 处获得主时钟的直接函数,在功能框 2466 处根据当前时刻和提供的速率计算从时钟的函数,功能框 2468 处计算直接函数,并在功能框 2470 处重新计算时钟唤醒,并在判别框 2472 处进行检测以确定是否方向改变了。若是,则所有唤醒在功能框 2474 处启动。若否,则对所有从时钟的处理在功能框 2476 处重复,处理结束于终端 2499。

新的时钟函数计算如下:

从时钟在同步到主时钟之前的函数:

$$T_{\text{slave}} = a_{\text{original}} * T_{\text{master}} + b_{\text{original}}$$

从时钟在与主时钟同步之后并有一新速率 a_{new} 时的函数。

$$T_{\text{slave}} = a_{\text{new}} * T_{\text{master}} + b_{\text{new}}$$

这里 b_{new} 为 $T_{\text{newslave}} = T_{\text{slave}}$ 即

$$b_{\text{new}} = T_{\text{master}} * (a_{\text{s}} - a_{\text{new}}) + b_{\text{original}}$$

Sync To()

Sync To()允许用户指定主时钟或时间源,速率以及偏移。处理与图 24 所示逻辑非常相似。该时钟新函数在 Sync To()情况下计算如下:

从时钟在同步到主时钟之前的函数:

$$T_{\text{slave}} = a_{\text{original}} * T_{\text{master}} + b_{\text{original}}$$

从时钟在与主时钟同步之后具有新的速率 a_{new} 和偏移 b_{new} 时的函数:

$$T_{\text{slave}} = a_{\text{new}} * T_{\text{master}} + b_{\text{new}}$$

一个时钟可与任何其他时钟同步。即使它们位于不同的时钟层级。

声音播放器实施

一个声音播放器目标包括一段数字化声音样本。它具有成员功能从而可以取出和存储声音样本。

一个声音播放器播放声流从而可在喇叭上听到该声音。通过从声音流中周期地读出数据并将其写入一个声音输出口便可实现上述功能。读出周期由一个数/模转换器(DAC)确定,DAC把数字声音样本转换成模拟电信号,该信号可由连到一个喇叭的音频放大器转换成声音。该DAC每秒转换N个数字声音样本。N叫做DAC的样本率。DAC有等待转换的声音样本的内部缓冲器。声音播放器在主处理器中执行,它把从一个声音序列读出的样本存入该DAC的缓冲器。当该缓冲器有一半为空时,DAC中断主处理器。声音播放器然后用从声音序列中读出的下一批声音样本充满该缓冲器。该处理示于图25中。

图26展示了声音播放器作为控制源的情况。声音播放器有一个作为另一时钟的控制源的时钟。这是通过确保声音播放器的时钟总是反映喇叭当时再现的样本的时间的方式而实现的。例如,若在时刻T,扬声器在播放声音序列起始部分的第88200个取样,且DAC和声音序列的取样率均为每秒44100个取样,则声音播放器的时钟必须是时间2秒。通过生成由DAC驱动的时间源可实现这一功能。每次DAC输

出一个信号指示缓冲区已由取样允满时,它会中断主处理器,主处理器然后将会把时间源的当前时间增加一个时间增量,该时间增加等于最后中断以来所转换的取样数量。这由以下决定:

$$\langle \text{增量时间} \rangle = \langle \text{最后一次中断的取样} \rangle / \langle \text{DAC 取样率} \rangle$$

该声音播放器把其软时钟和 DAC 时间源同步。其线性函数符合下式:

$$T_{\text{audioplayerclock}} = 1 * T_{\text{DACtimesource}} + b$$

这里 b 被设定以致当序列播放开始时, $T_{\text{audioplayerclock}} = 0$ 采用时钟的 SetTime 功能可实现这一功能。

音频播放器作为受控源

声音播放器可与一外部时钟同步。在这一情况下,声音播放器必须确保扬声器再现的样本与外部时钟。例如,若在一给定时刻外部时钟目标的当前时刻是两秒,则在声音序列中的第 88200 个样本会在该时刻由扬声器再现,假设声音序列的取样率为 44100。该处理实现如下:声音播放器等待来自 DAC 的中断,指示需要更多声音数据。该播放器然后检查其时钟的当前时刻,确定是否需要加速或降低播放速率或跳到声音序列中的新位置,以便确保正确的样本从扬声器播放。然后使用数字信号处理算法在输出取得之前把声音序列取样率转换成 DAC 的取样率。

图 27 展示了根据本发明最佳实施例与控制一个声音播放器有关的详细逻辑。处理始于 2700,在该处进入等待状态

直到一个 DAC 中断发生。当 DAC 中断发生时,控制去往功能框 2710 以获取时钟时间。然后在功能框 2720 处计算与该 DAC 速率有关的时钟速率,并在 2730 处进行测试以确定时钟速率是否太多。若是,则将时钟速率设定为最大,若否,则在功能框 2750 处计算需要的位置。功能框 2750 需要更多解释,因为其处理涉及更多。即是声流理想位置的计算。这一位置确定了播放器将输出的下一样本。因为 DAC 引入的延时,这一值的计算是不普通的。当播放器把一个样本输出到 DAC 时,它进入 DAC 的内部缓冲区。直到所有在 DAC 缓冲区里的样本都空了的时候,扬声器才停止再现样本。幸运地,该延时可按下式计算:

$$\text{delay}_{\text{DAC}} = \langle \text{Samples in internal buffer}_{\text{DAC}} \rangle / \text{Sample Rate}_{\text{DAC}}$$

声音序列取样将以时钟速率(Clock Rate)确定的速率输出,这来自于与 DAC 时钟位置相关的外部时钟位置。为确定理想位置,DAC 延时被减去,但是此之前必须被转换成声流的时基。这是按以下方式完成的:

$$\text{delay}_{\text{audiosequence}} = (\text{delay}_{\text{DAC}} * \text{ClockRate})$$

声流的理想位置则按下式计算:

$$\text{desiredPosition} = \langle \text{Position of next sample in sequence} \rangle - \text{delays}_{\text{audio sequence}}$$

然后,在功能框 2760 处比较需要的位置和实际位置以确定是否其太偏离实际位置。若是,则在功能框 2770 处查找声音序列到需要位置。在任何情况下,在功能框 2780 处取样

图形播放器实施

一个图形序列目标是一个图形目标集。图形序列中的图形目标可被数字化成为视频帧,提供动画片的帧,或任何其他与时间相关的图形目标。一个图形序列具有成员功能,即可以访问存贮和检索用的各个单独的图形。每一图形有一期间;因此,图形有一等于分量图形目标期间总和的时间。图形序列还提供把序列的期间内的时间映射到一特定图形目标的成员功能。图 28 展示了日出的图象序列示例,其中每个图形目标的期间示于其下方。

一个图形播放器通过其图形输出端口播放一个图形序列,该图形端口连到一个图形输入端口。该播放器有一可以驱动其他时钟(控制源)或被驱动(受控源)的软件时钟。该播放器根据其时钟时间顺序通过图形序列的图形目标时钟被限于从零到等于被播放的图形序列时间的时间的范围之内,播放器延时直到当前帧的期间已在时钟上消逝。然后播放器检测时钟上的当前时刻,获得映射到那一时刻的序列的图形目标,并将该图形目标写入到输出端口。

播放器在选择将被写入到输出端口的图形目标时总是用当前时刻。这样便在低性能时提供自动降级。若写入图形目标用时超过其时间,至少会漏掉一帧。时钟方向的变化也会有类似情况发生。与图形播放器相关的详细逻辑示于图 29 的流程图中。处理始于 2900 并立即去往功能框 2910 以进入延迟直到下一图形目标的时间。然后在该延时之后,在功能框 2920 处获得时钟时间,图形目标在 2930 处写入一个

输出端口,并在功能框 2940 处进行测试以确定是否已调用停止成员功能。若是,则处理结束于 2960。若否,则在判别框 2950 处进行测试以确定是否是最后一帧且时钟是否受控。若否,则控制去往功能框 2910。若是,则处理结束于终端 2960。

注意若播放器的时钟与另一时钟同步,则显示循环继续执行直到播放器确实停止。但是,若时钟不同步,则在序列的最后图形被写入时退出循环。

MIDI 播放器实施

一个 midi 序列目标是一个 MIDI 事件集。其成员功能控制播放以及访问 MIDI 事件的时序轨道。一个 MIDI 序列和 MIDI 追踪格式示于图 30。

一个 midi 播放器通过其输出端口播放一个 midi 序列,该输出端口与任意输入端口相连,这些输入端口包括在 MIDI 合成器上用于最终输出的 MIDI 驱动器的输入端口。该播放器跟随软件时钟从而在事件成为当前事件时将其输出,该处理可良好地在正向和反向时间流中完成并在主处理器落后时考虑降级决策。图 31 展示了通过一个驱动器为一个合成器播放一个序列的 MIDI 播放器。

MIDI 播放器可作为同步的控制源或受控源,其取决于所采用的由其基本分类——时基媒体播放器(Time-Based Media Player)——继承而来的 SyncTo 和 SyncToSelf 成员功能。实施细节不改变并跟随一个具有两个特殊事件的简单时序事件循环。第一个特殊事件处理时间转换方向,通常,播放

器通过延时直到把下一事件成为当前事件时进行工作。在延时结束时,播放器对时间进行检测以确定是否有一特定事件发生。第一特殊事件发生于延时是较早的时候。这一情况由下面的主循环流程中的第一判别框表示。在检测时,播放器将改变方向,所以“下一个”变成为与其原来相反。第二个特殊事件与处理器超载有关。当主机不再保持,应用降级决策。播放器在到达序列终端时停止于正向情况下的最后事件或在反向情况下的第一事件。

图 32 展示了 MIDI 播放器详细逻辑的流程图。处理始于 3200 并立即去往功能框 3210 以进入一个直到下一 MIDI 事件的延时。当下一事件准备好要被执行时,在功能框 3220 处获取时钟时间,在判别框 3230 进行检测以确定是否该事件是早。若是,则方向改变且控制去往功能框 3210。若否,则在功能框 3242 处进行另一检测以确定该事件是否晚。若是,则应用降低以赶上,且控制去往功能框 3260,并在功能框 3270 处进行测试以确定是否最后事件已完成且时钟目标不是受控源。若最后事件已执行且时钟不是受控的,则控制去往功能框 3210 以处理下一事件。

显示屏幕

图 33 到 37 示出了根据本发明的最佳实施例的显示。图 33 展示了一个空的桌上显示,用户被其表达从而开始定义软件时钟并将其与一个多媒体目标同步。图 34 展示了根据本发明最佳实施例的选择时钟目标定义显示。用户从信息 3410 的选项屏中选择时钟目标定义。尤其是,时钟选项屏项

3420 所示。图 35 展示了根据本发明的最佳实施例用图 34 所示的选项屏选择限定的各种时钟目标和多媒体目标。一个移动条目标 3510, 微调/快移控制器 3520, 时钟目标 3530, 以及一个动画多媒体目标 3540 被展示, 就象它们出现在一个实际的多媒体表示中一样。

图 36 是根据本发明的最佳实施例连接到一起的多个时钟目标以及多媒体目标。该连接由一个光标把一个几何图形例如一个线段进行橡皮带生成的方式而生成, 从而可把一个时钟目标 3610 连到另一时钟目标 3620, 或多媒体目标 3630 和 3640。图 37 展示了根据本发明的最佳实施例与一个声音目标同步的视频目标。视频目标 3710 与音频时钟目标 3720 同步以控制由动画多媒体目标 3730 表示的显示以及有关的多媒体音乐表示。

本领域的一段技术人员可依据本发明的精神对以上实施例进行变型和变形, 本发明受权利要求的保护。

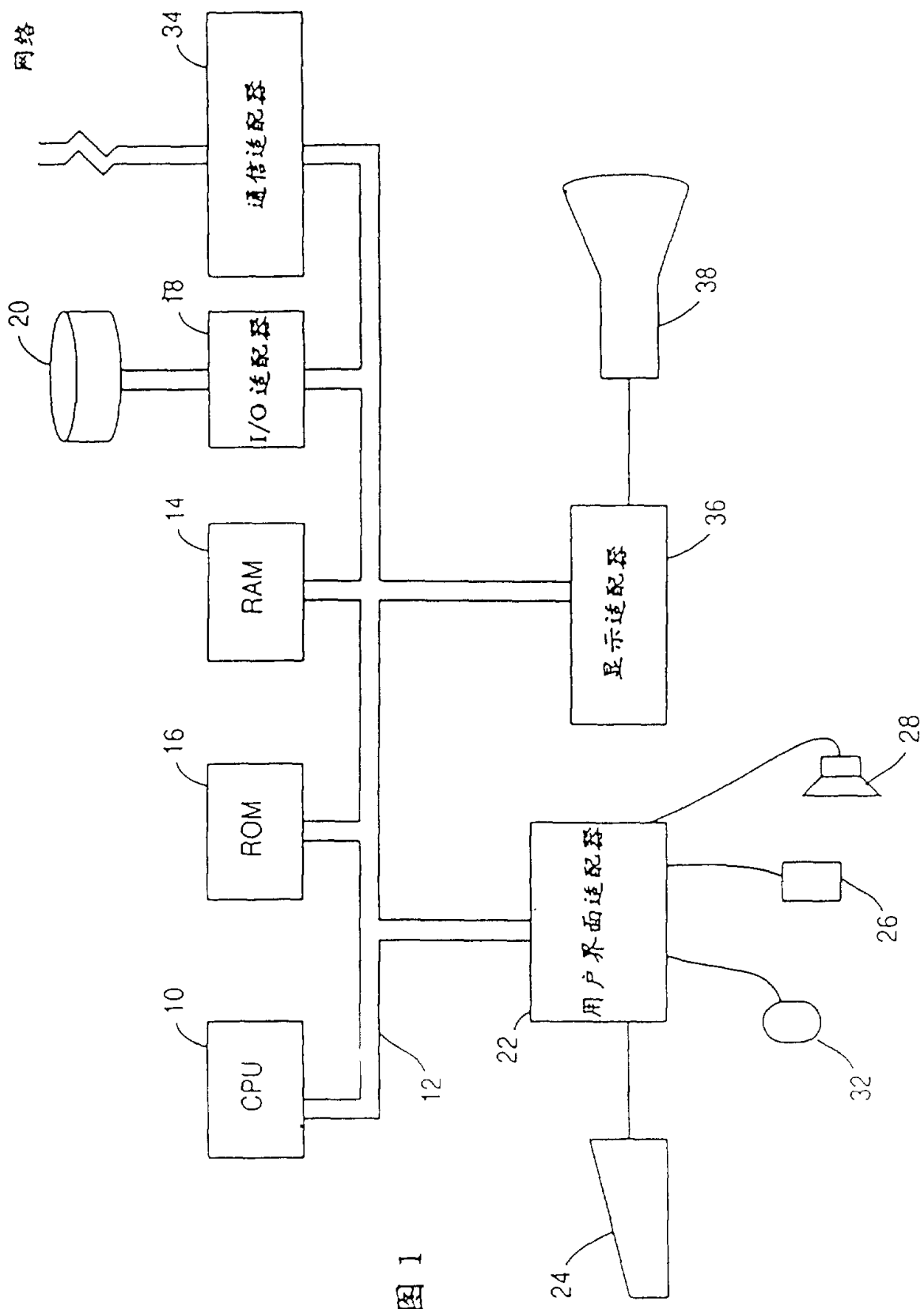


图 1

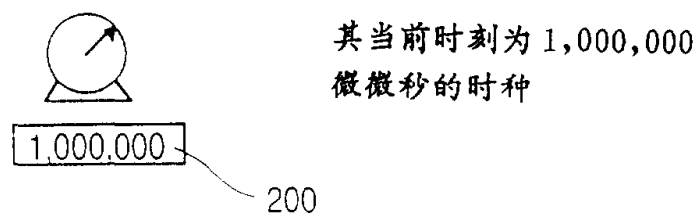


图 2

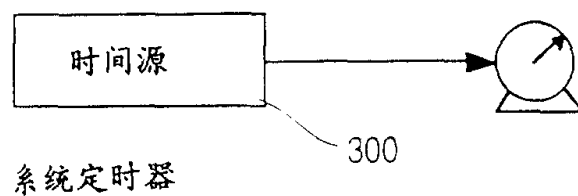


图 3

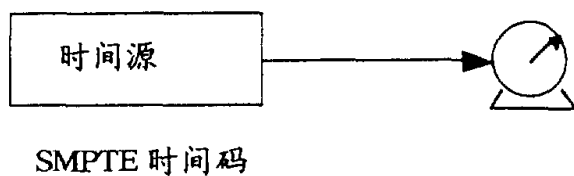


图 4

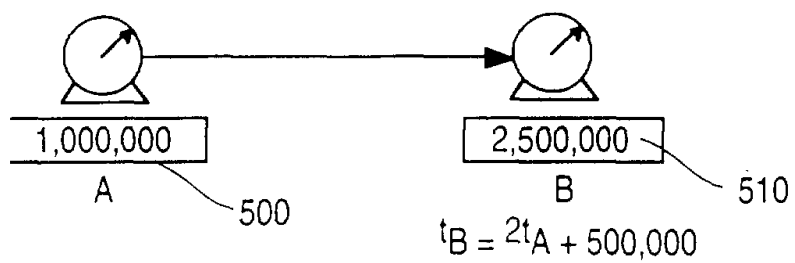


图 5

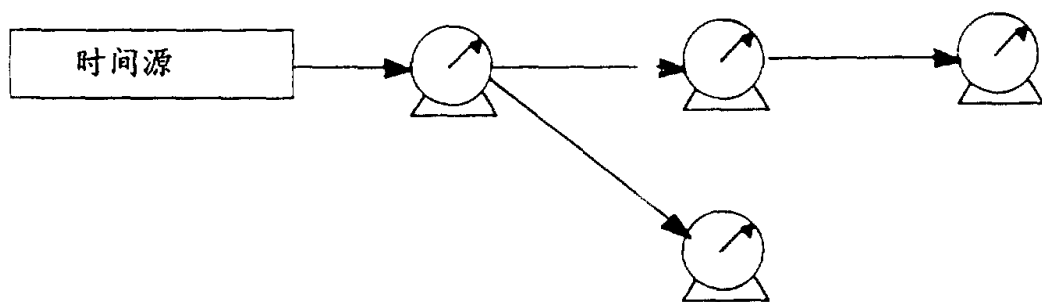


图 6

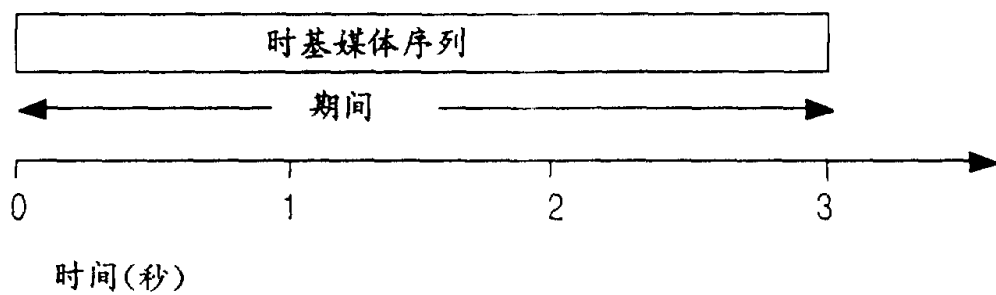


图 7

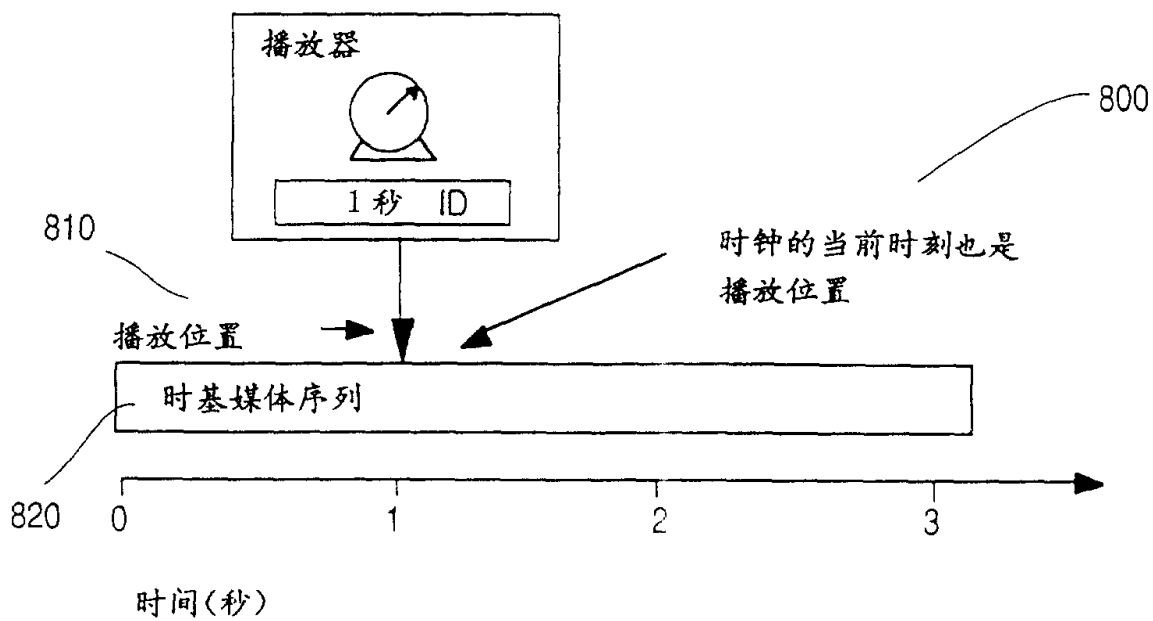


图 8

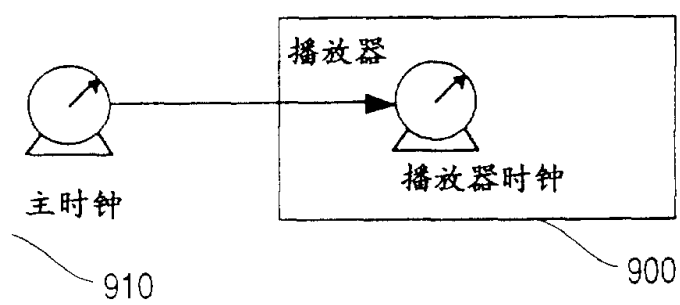


图 9

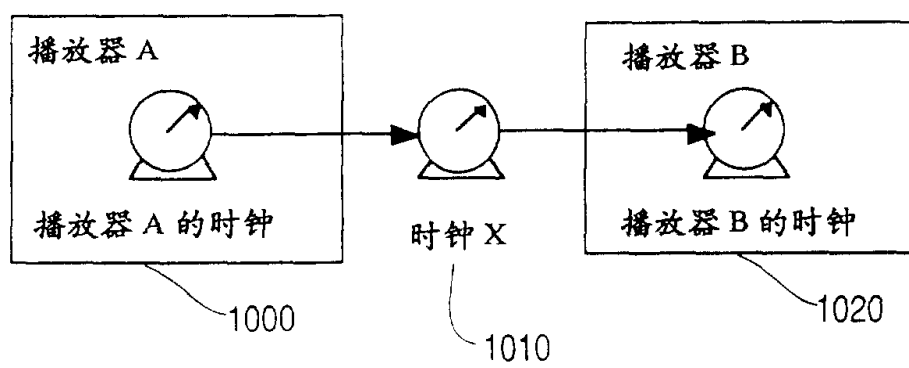


图 10

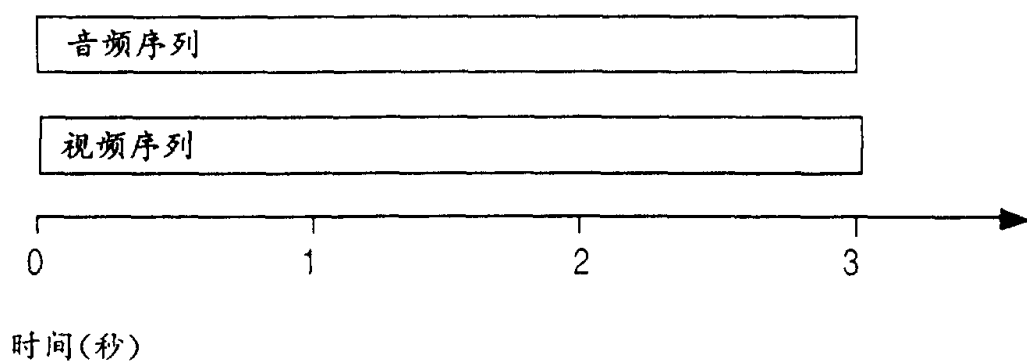


图 11

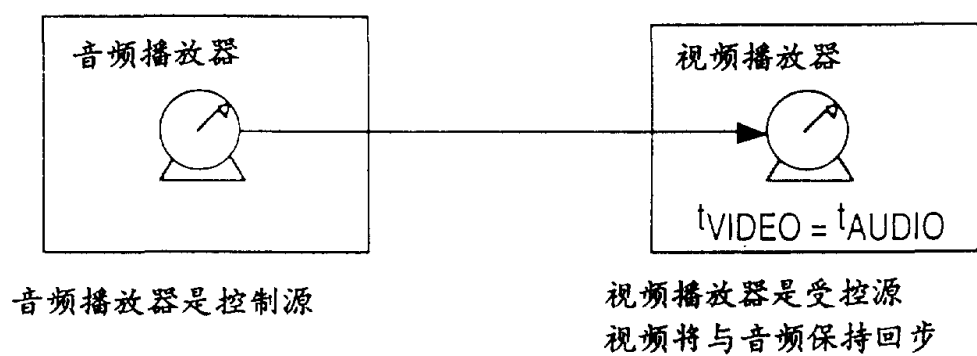


图 12

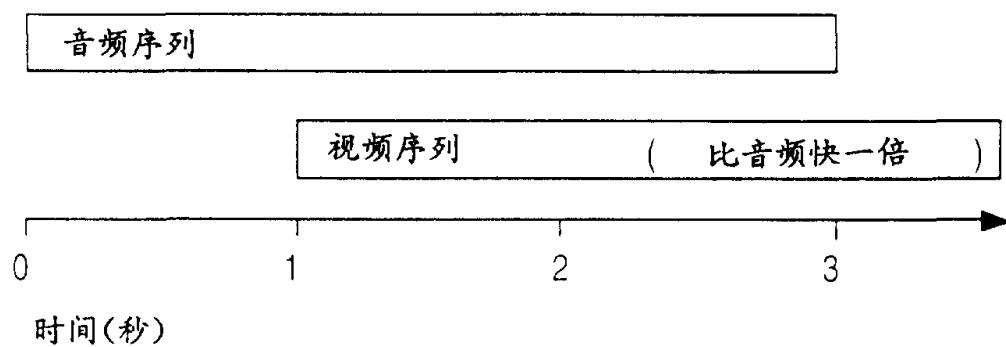


图 13

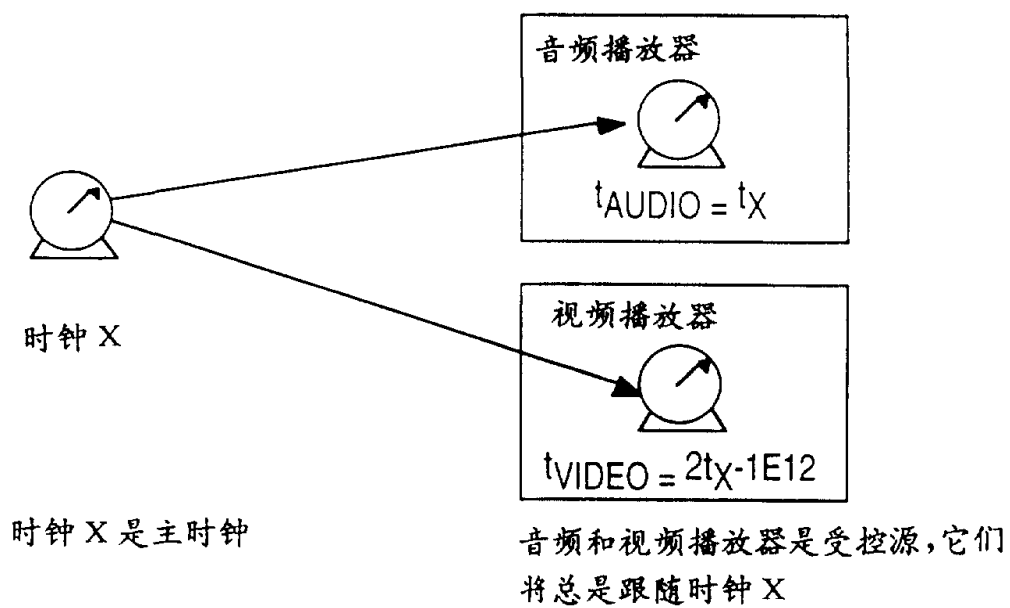


图 14

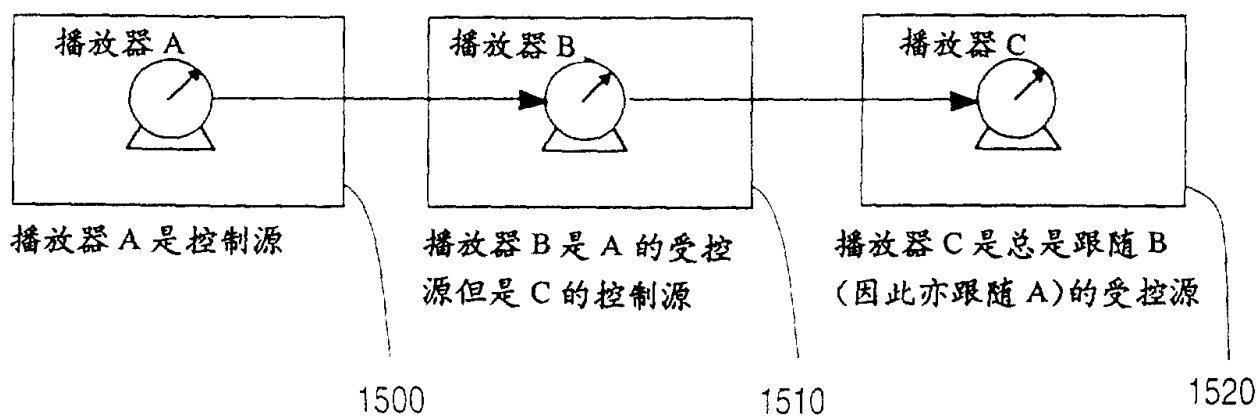
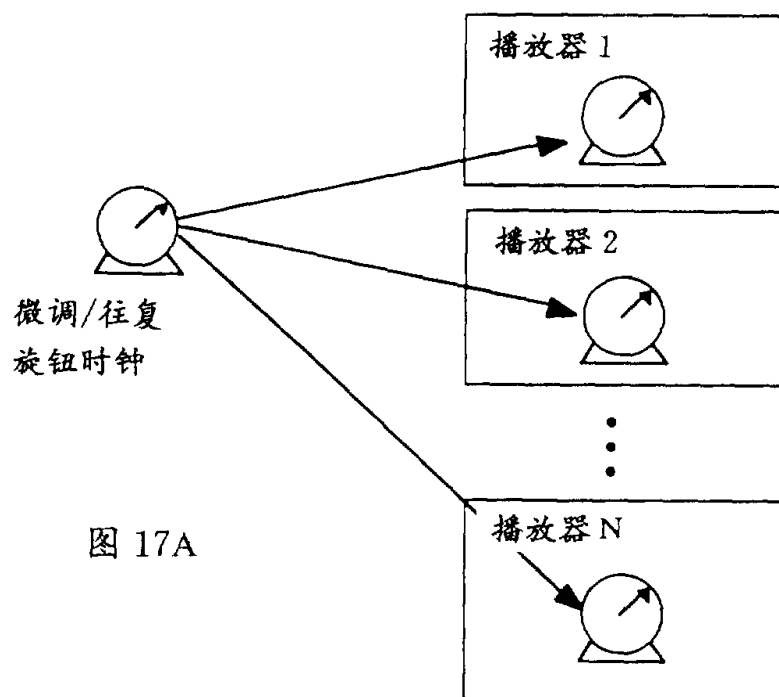
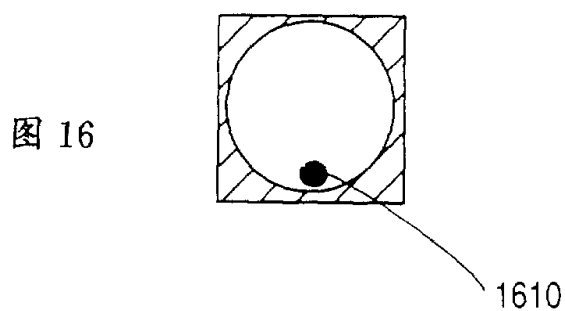
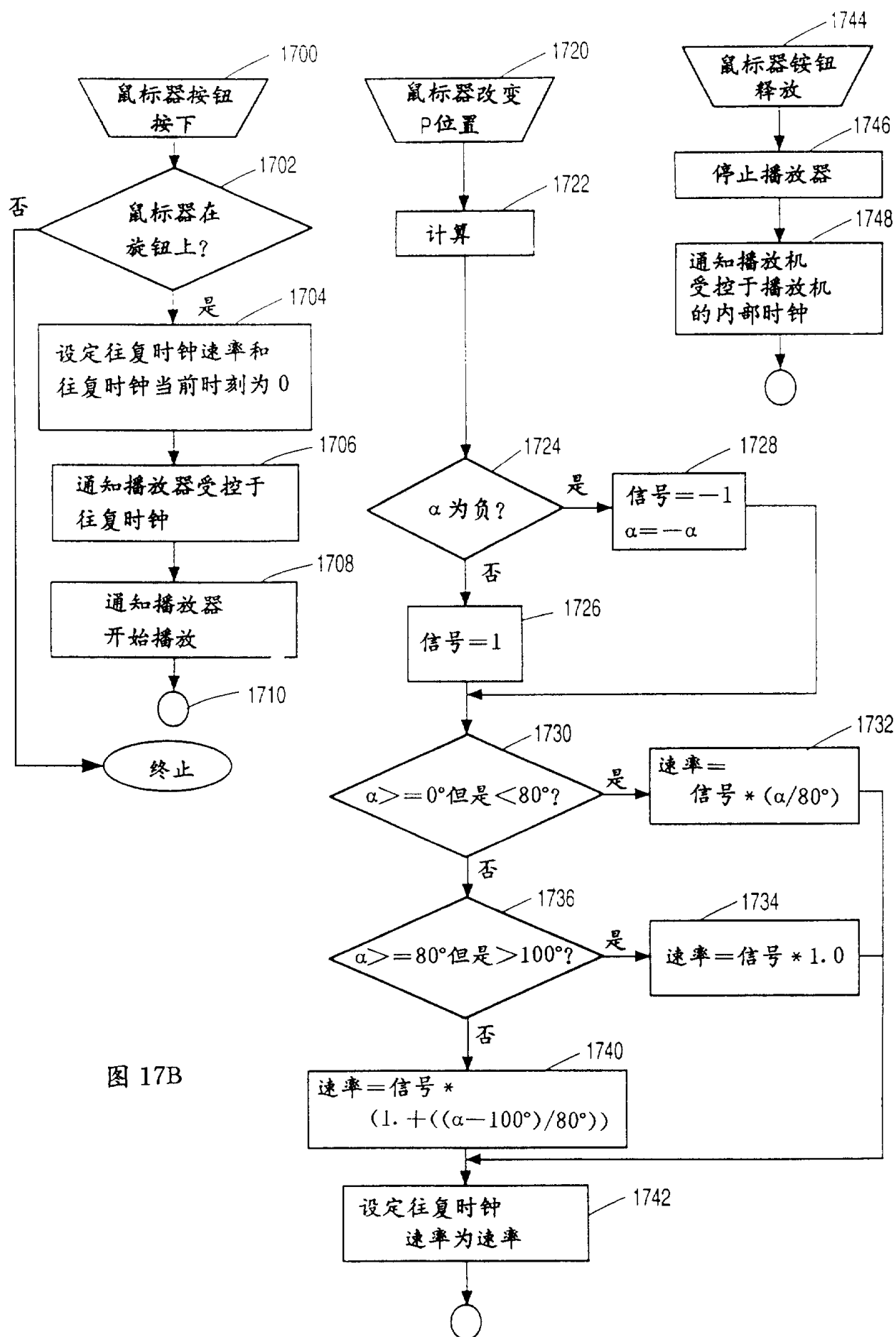


图 15





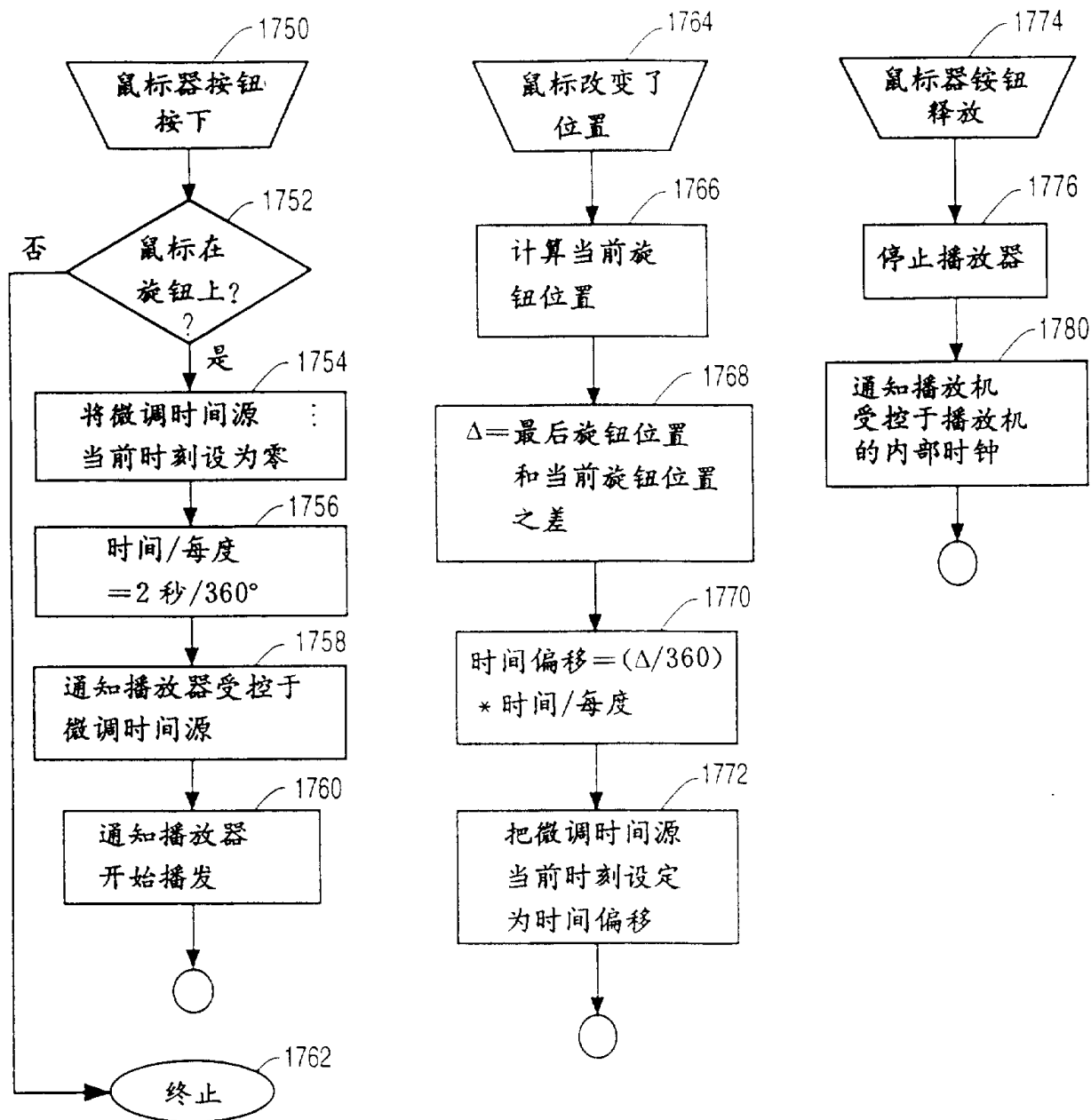


图 17C

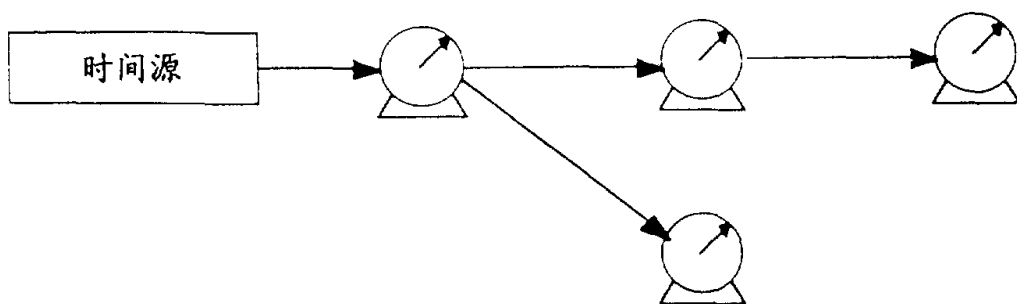


图 18

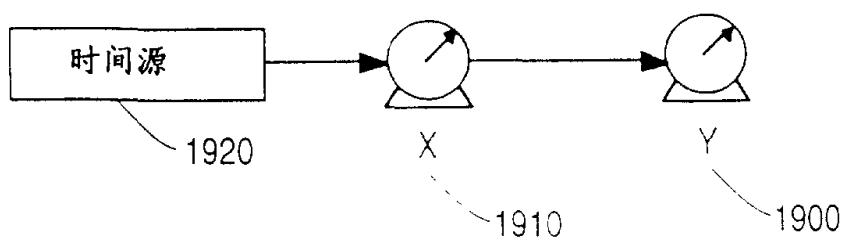


图 19

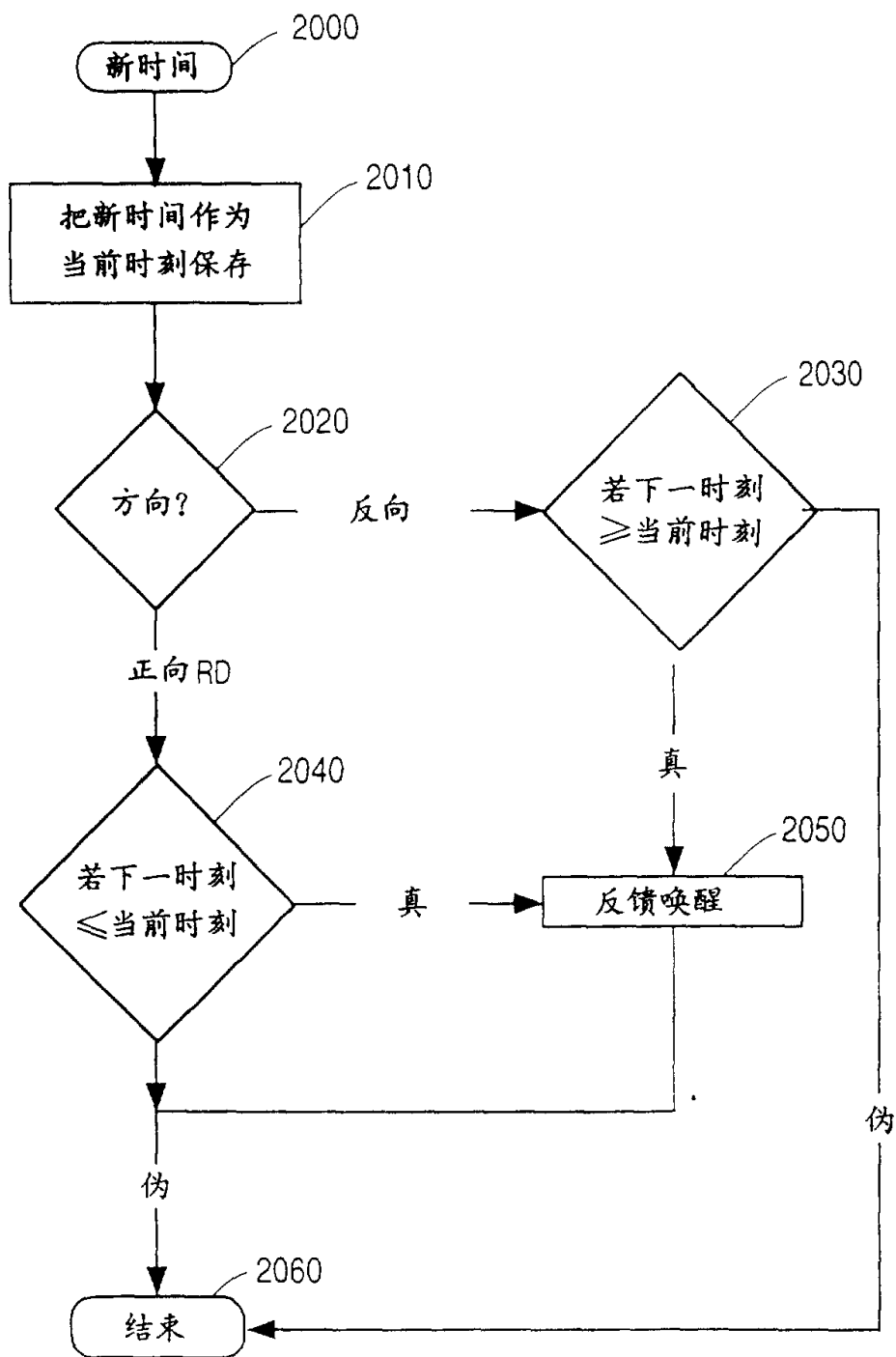


图 20

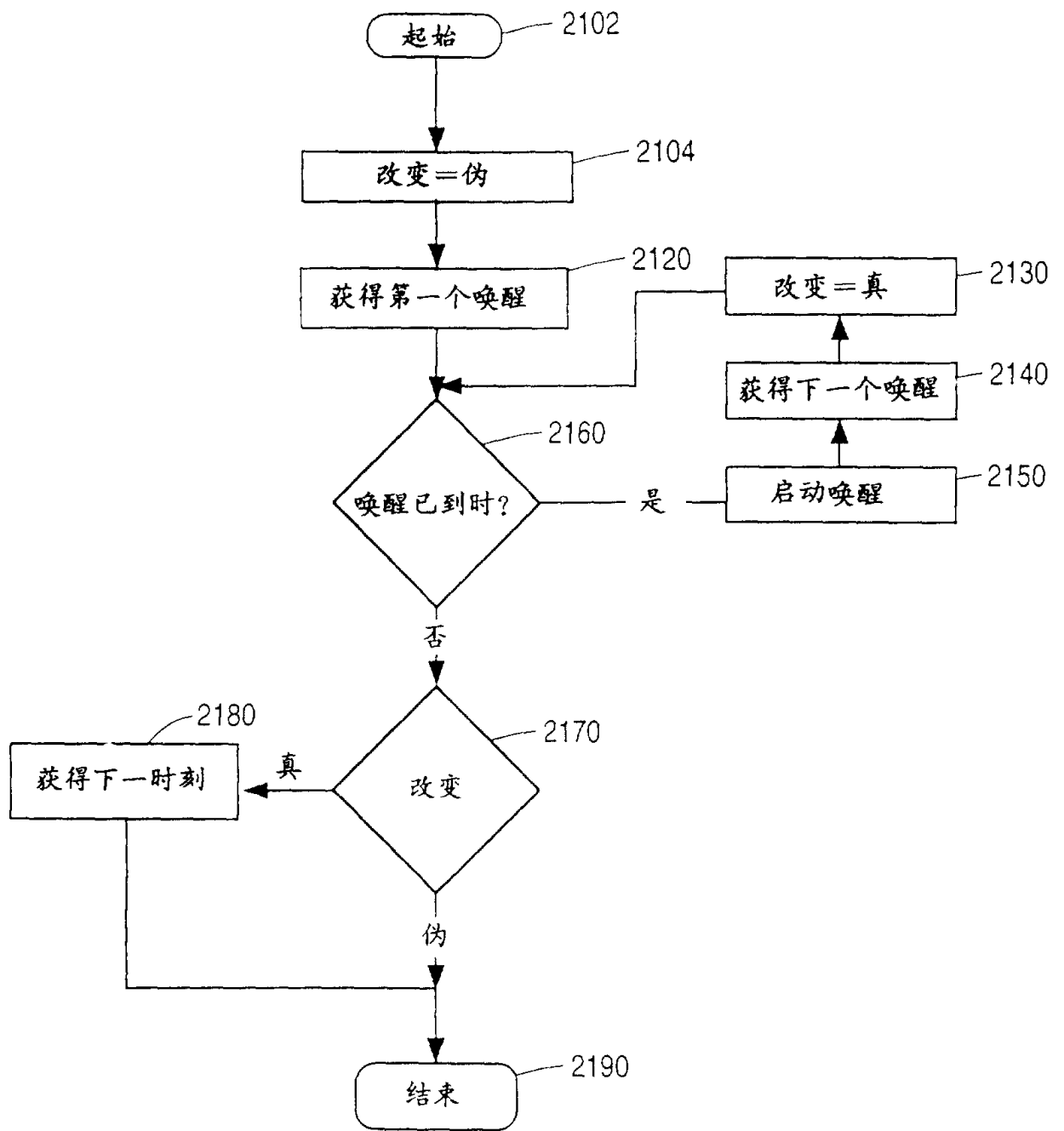


图 21

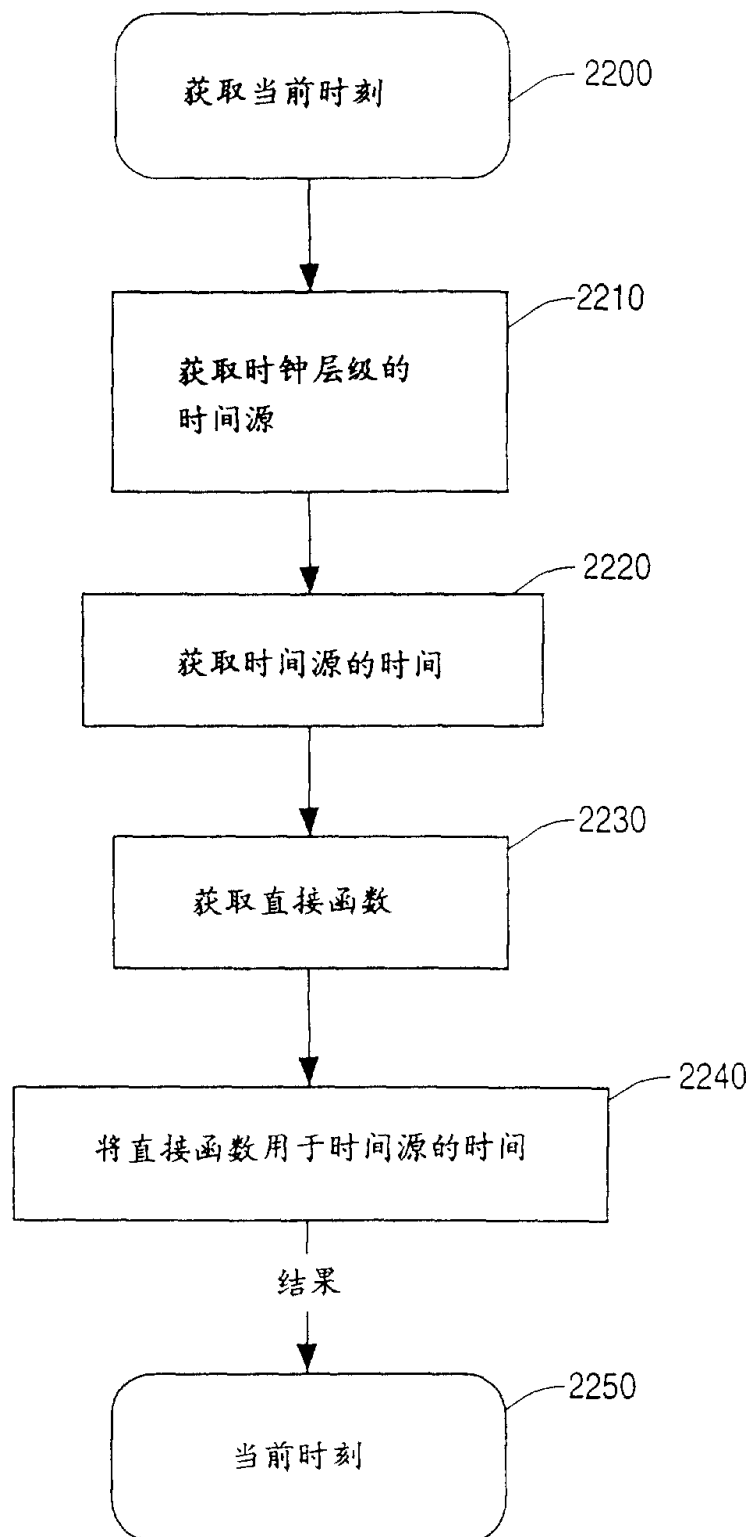
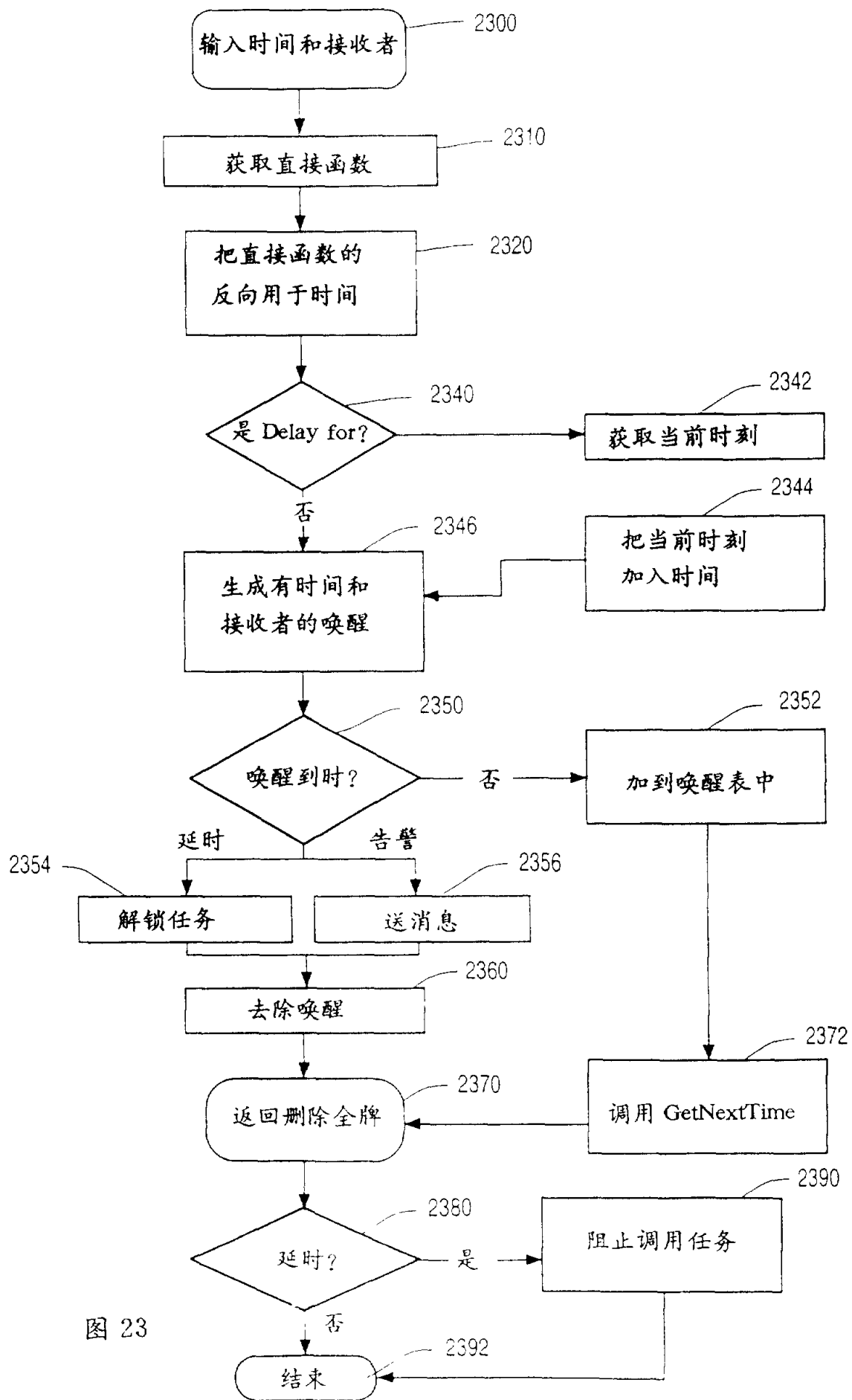


图 22



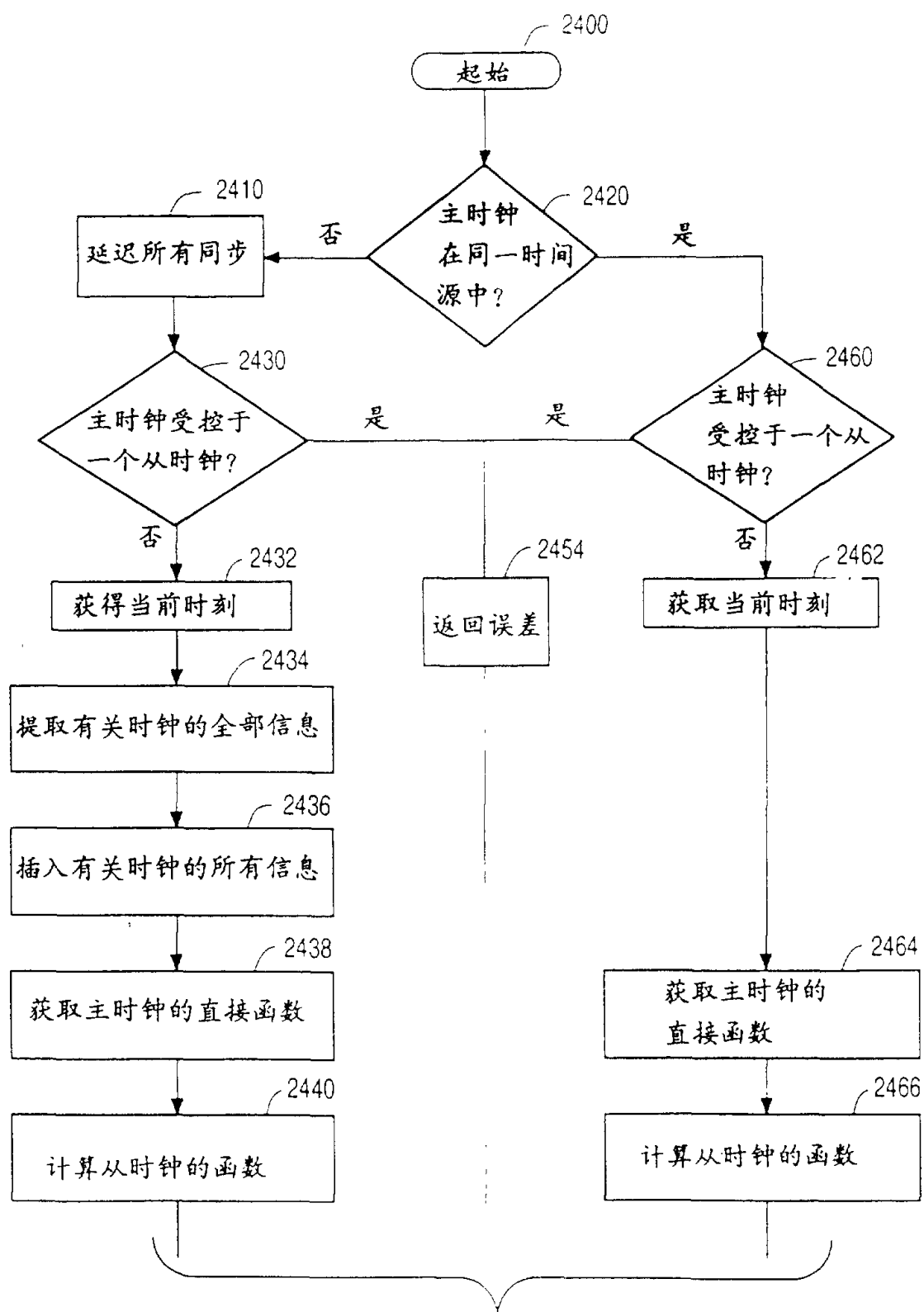


图 24A

到图 24B

来自图 24A

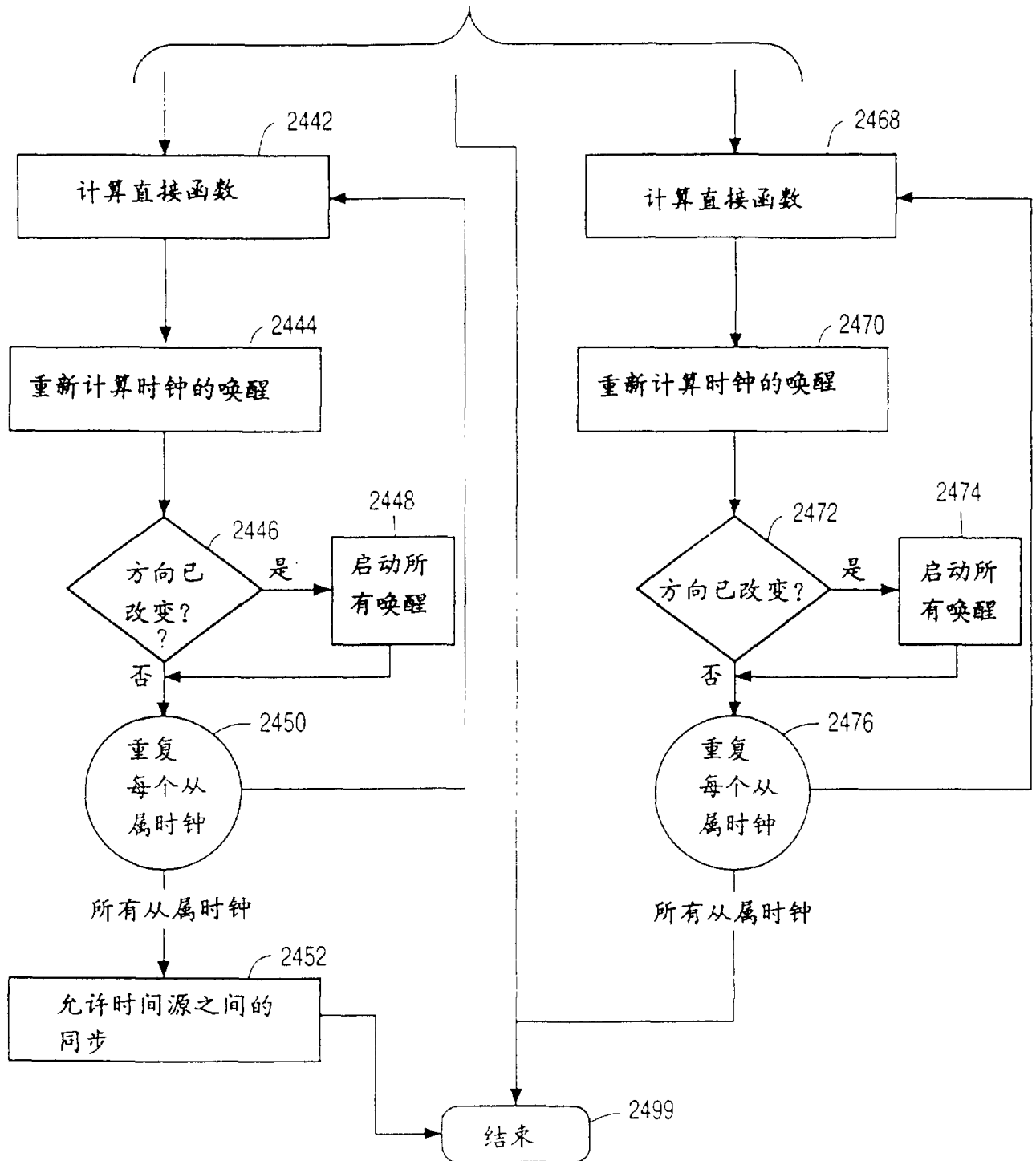


图 24B

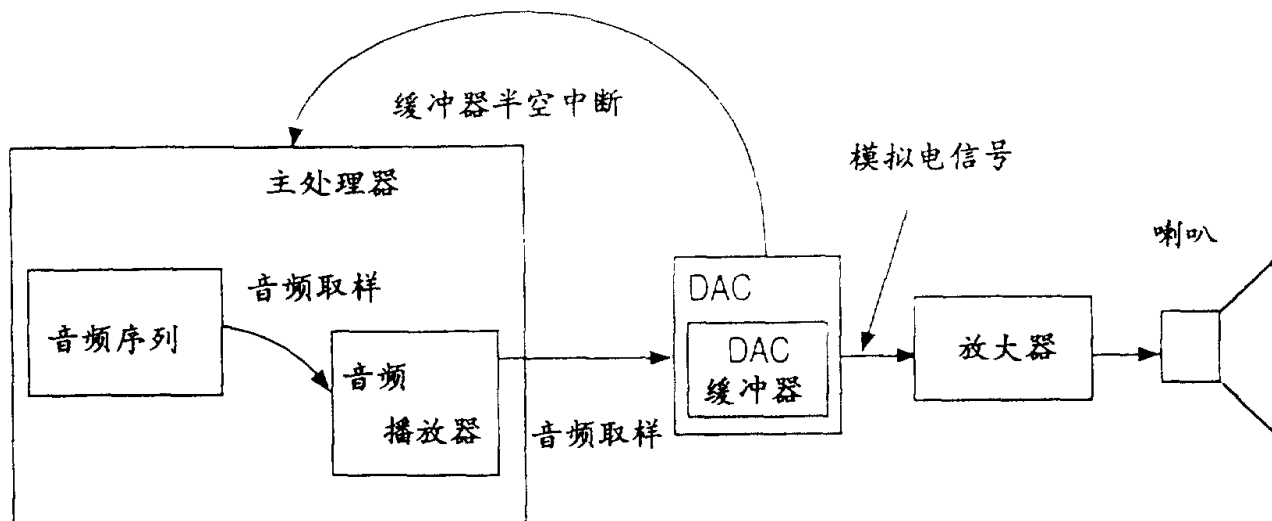


图 25

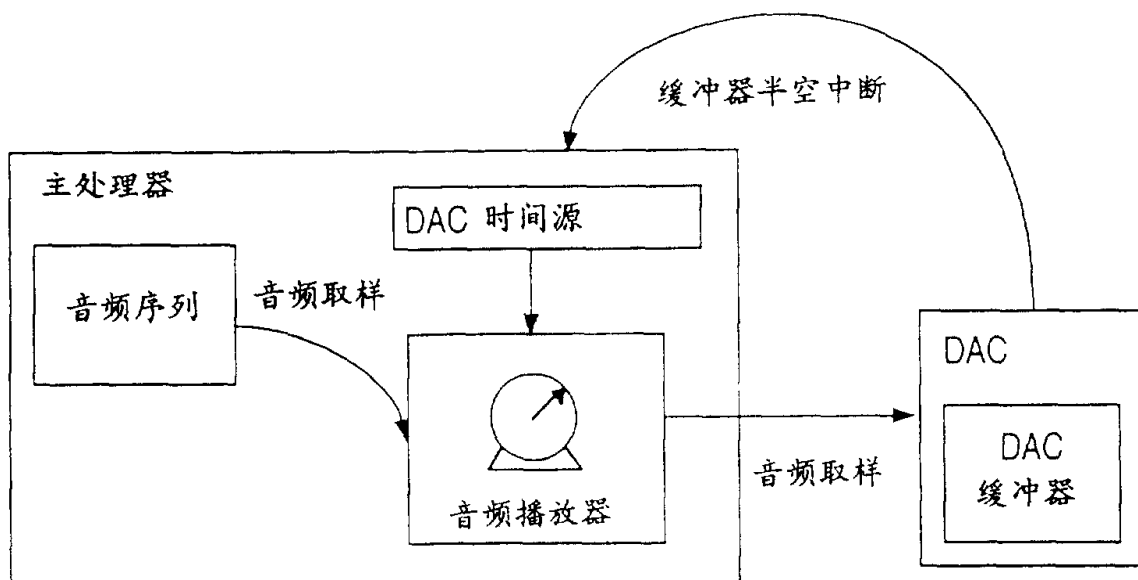


图 26

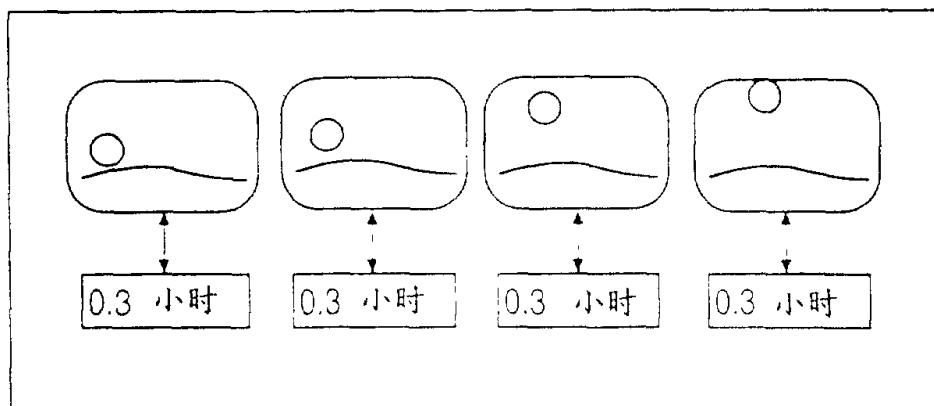


图 28

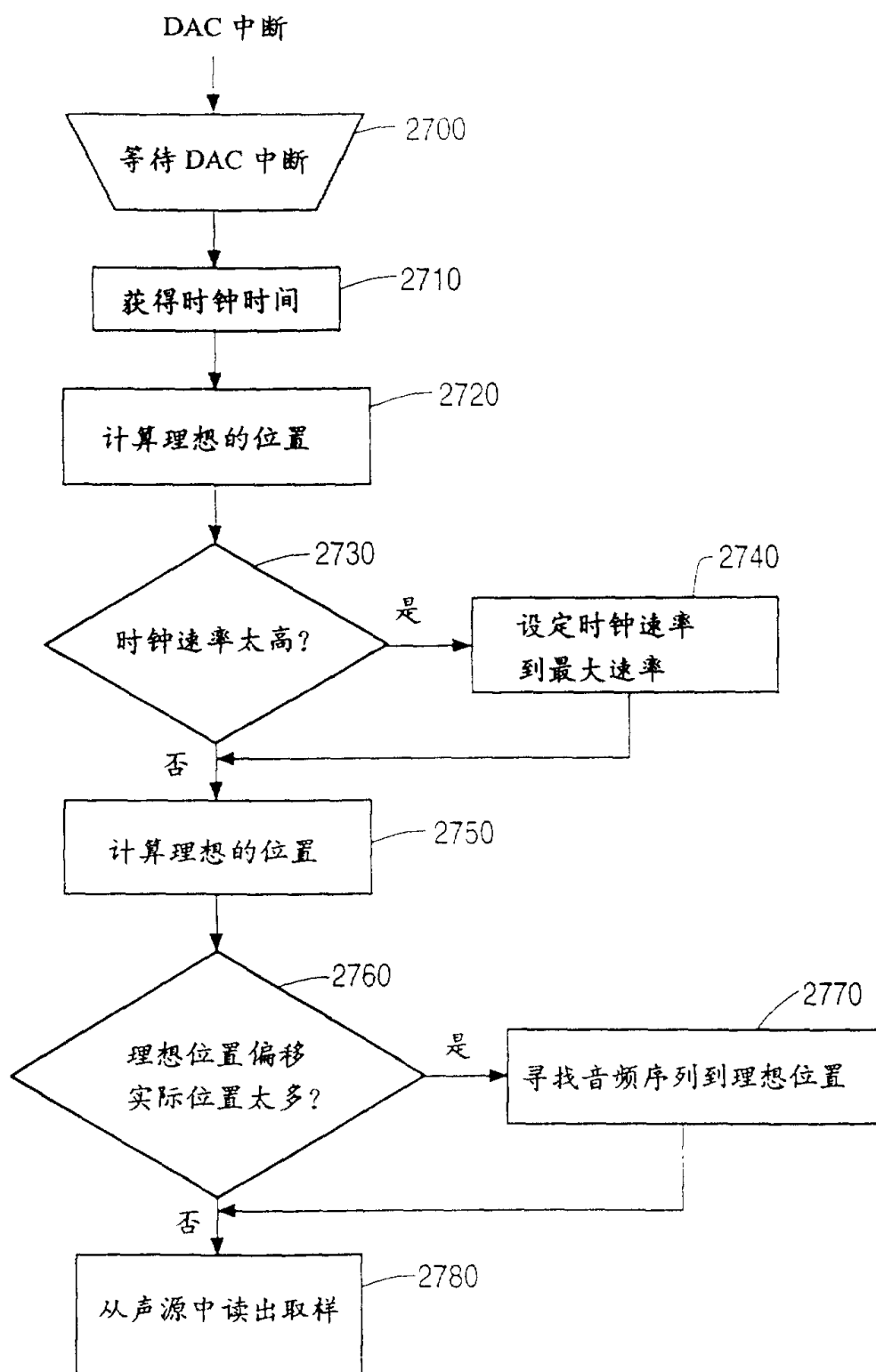


图 27

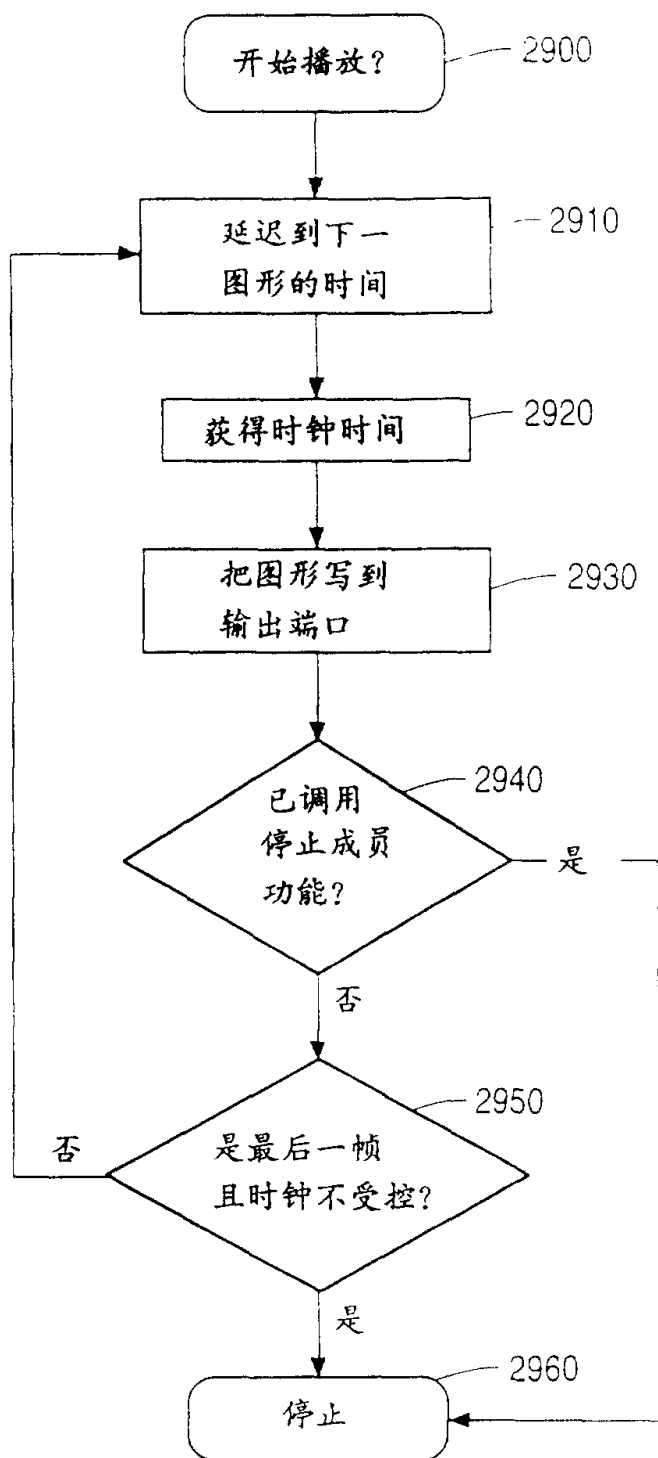


图 29

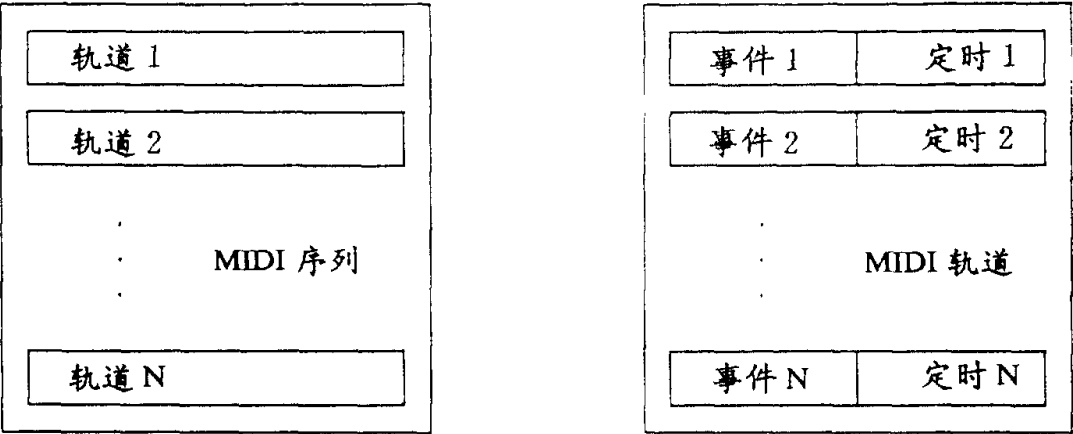


图 30

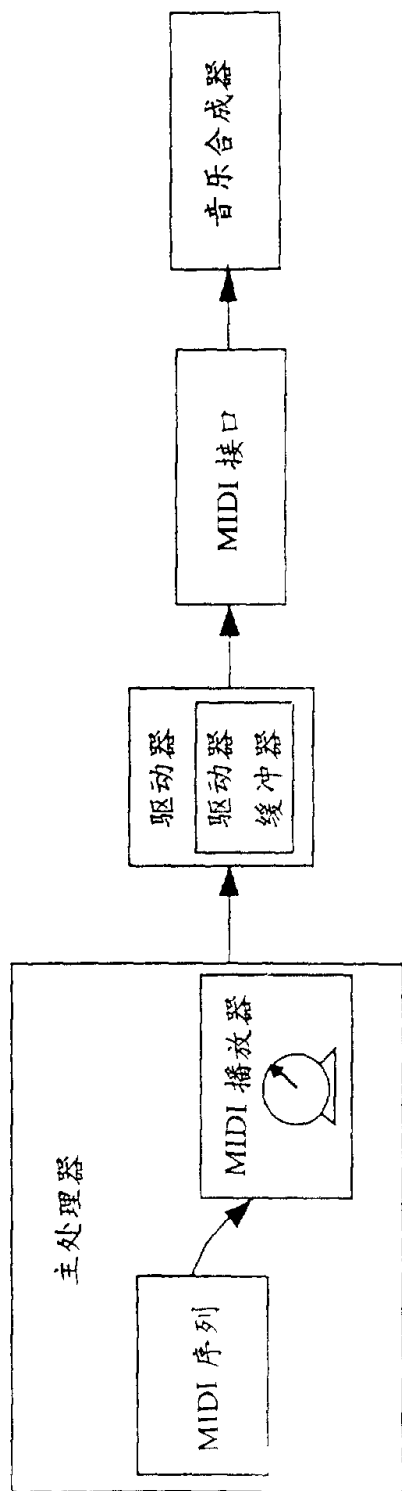


图 31

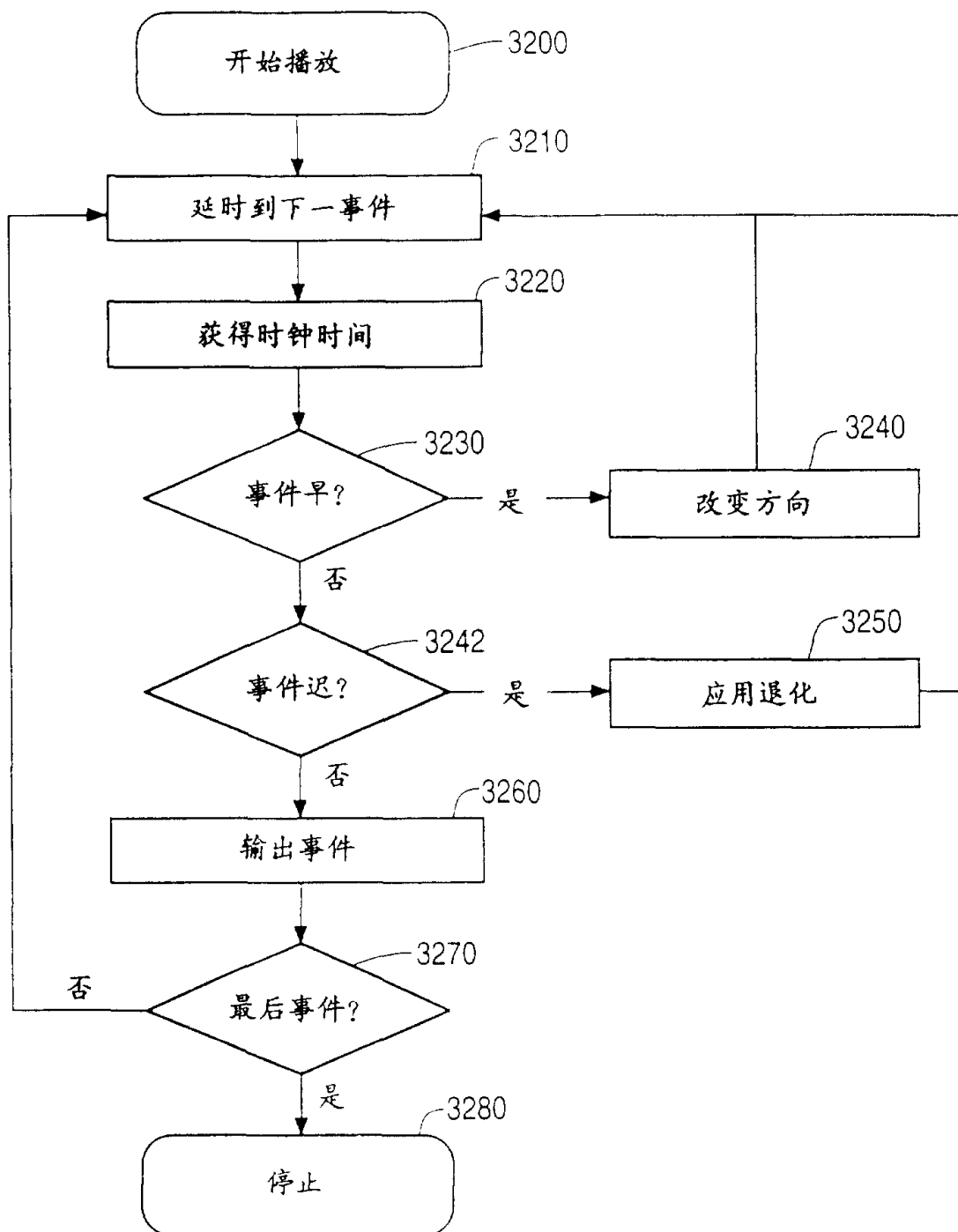


图 32

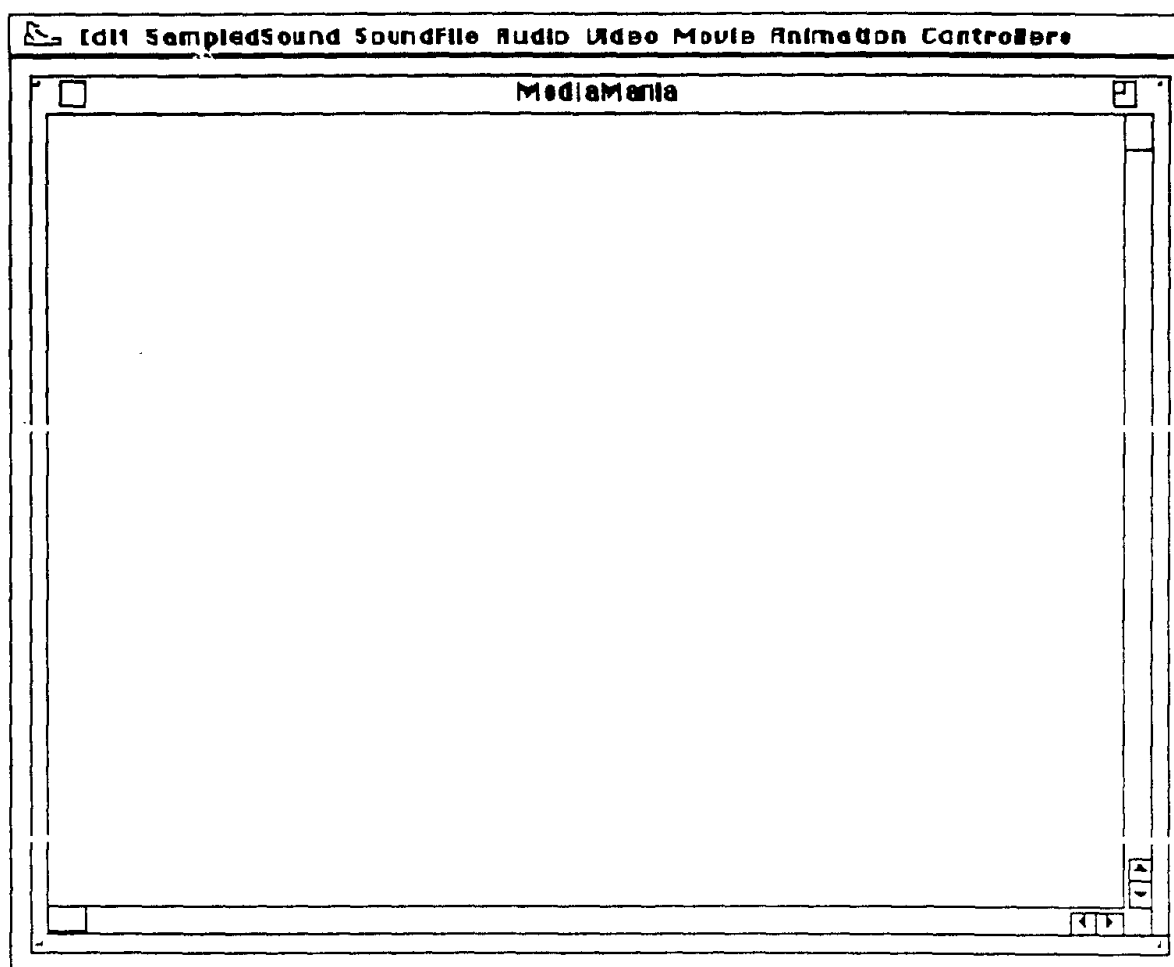


图 33

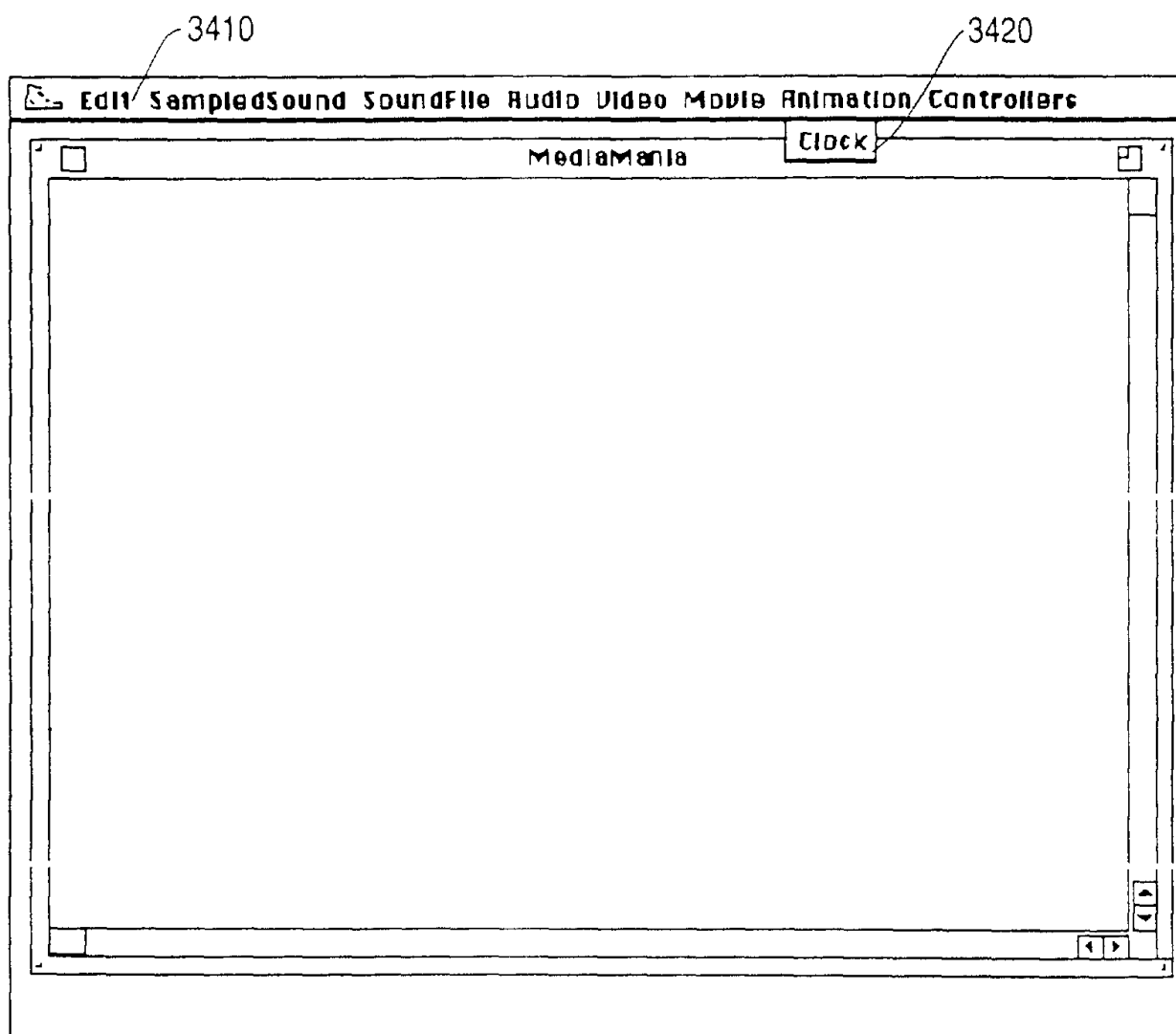


图 34

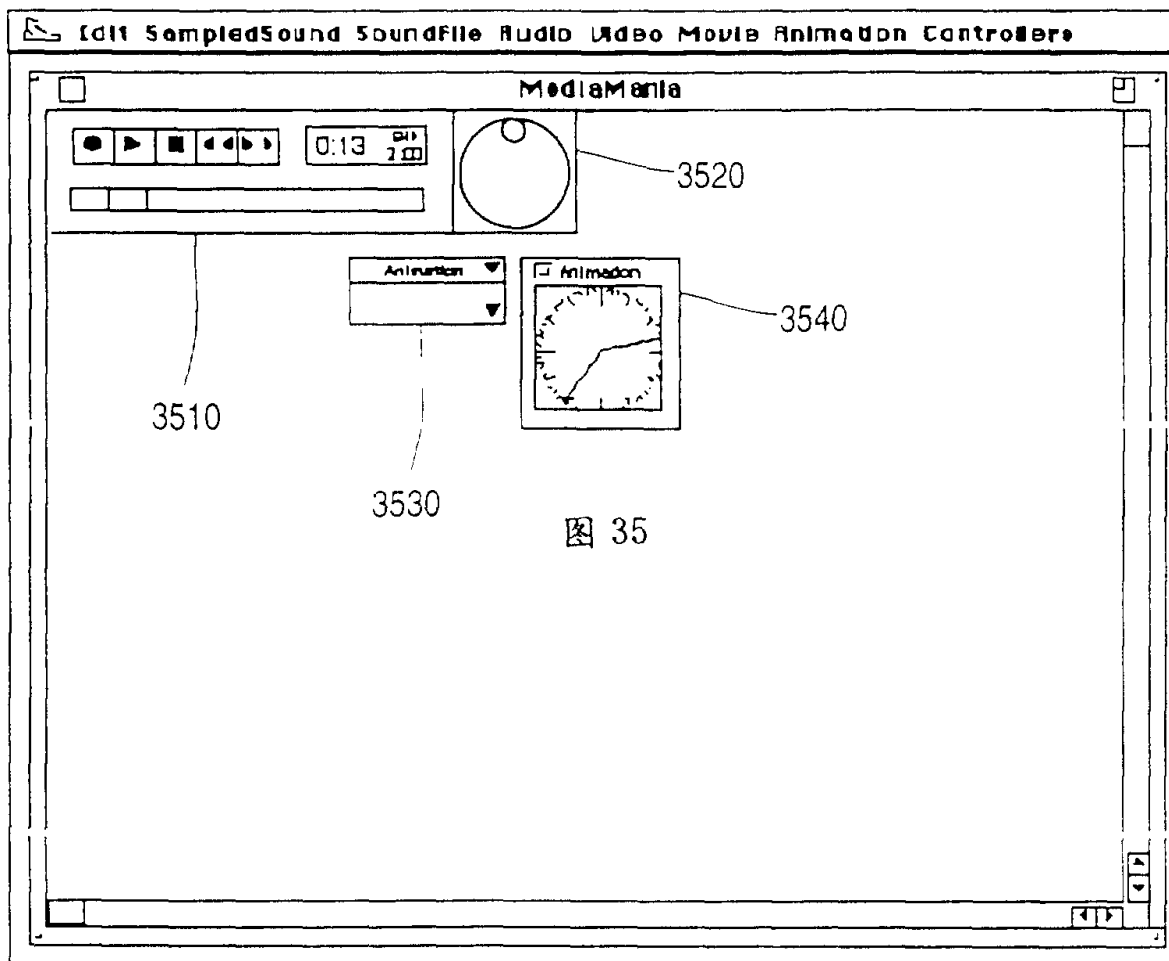


图 35

图 35

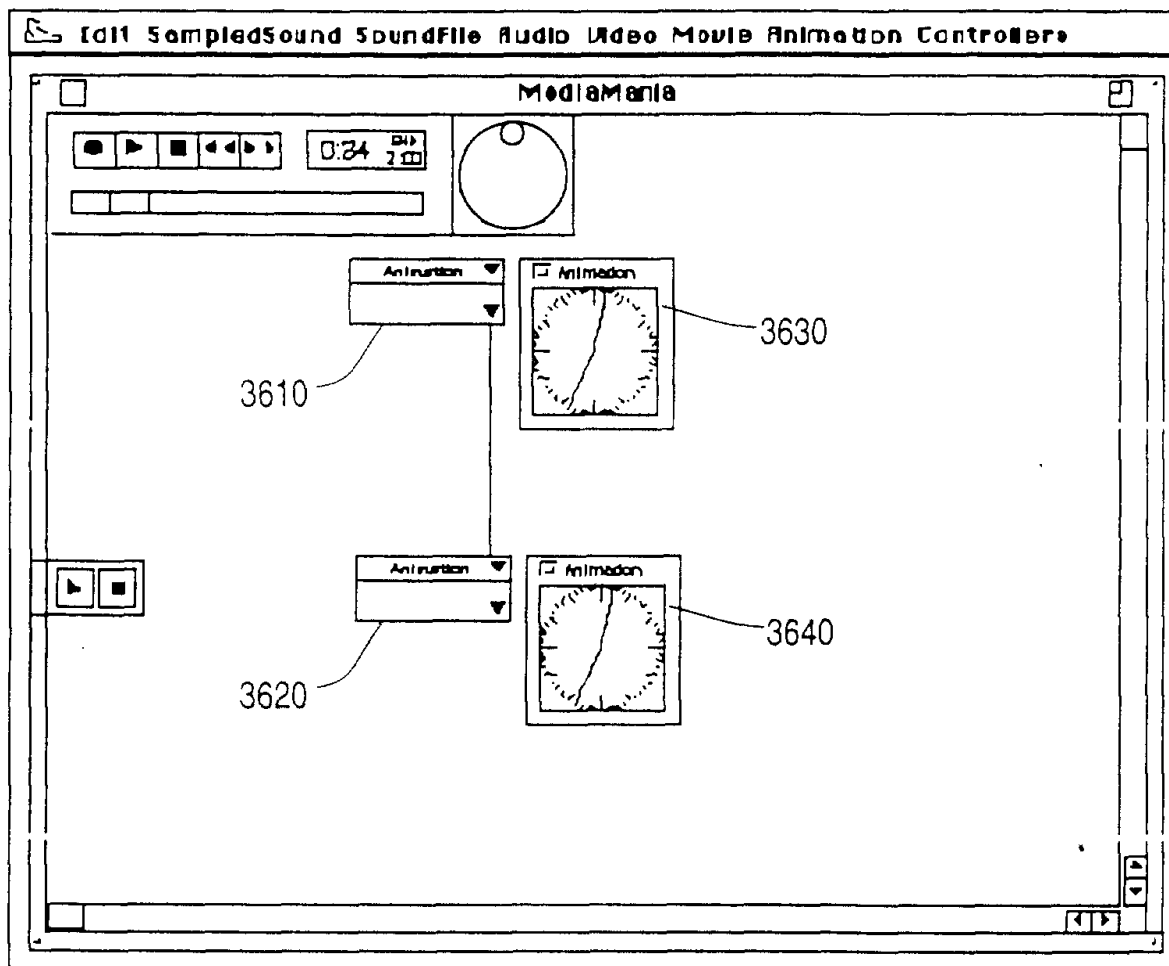


图 36

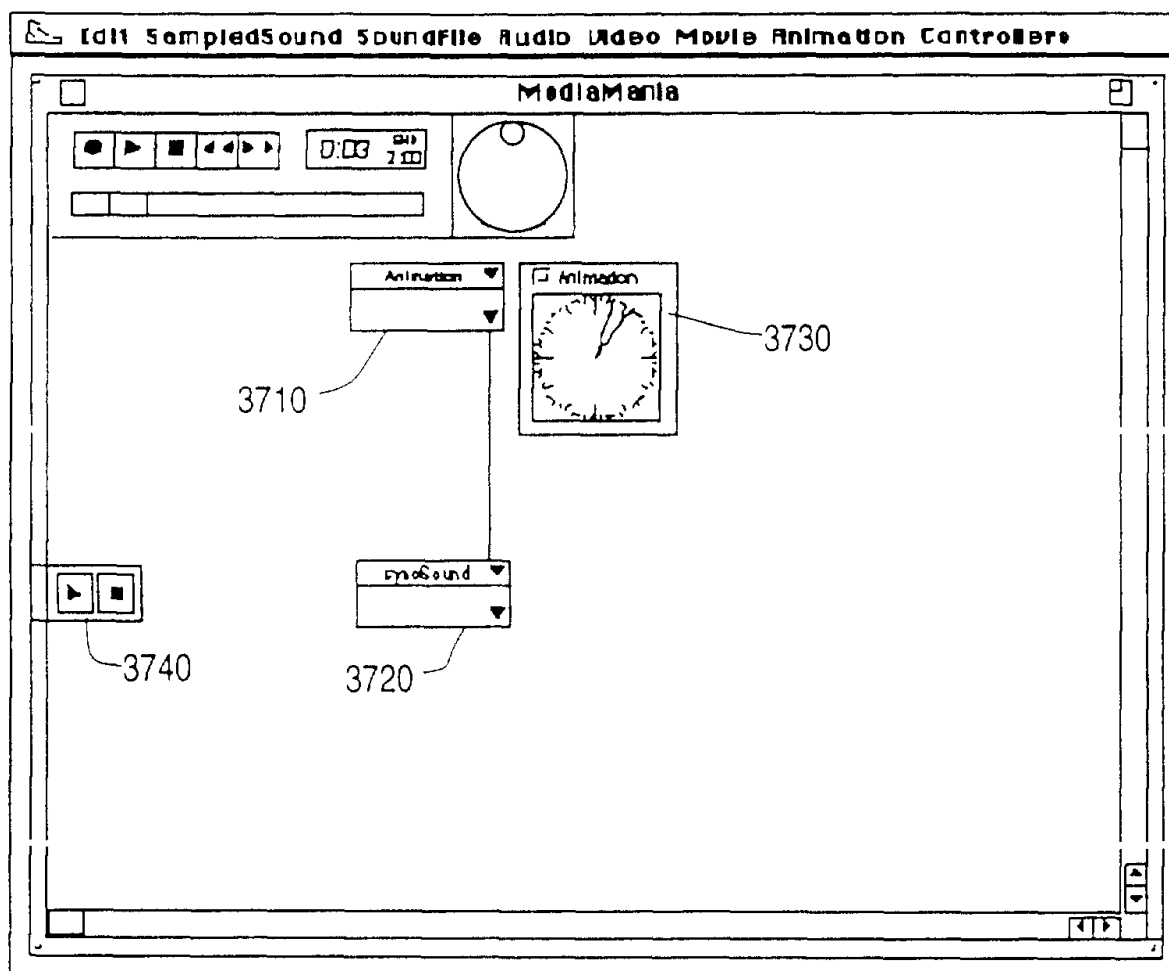


图 37