(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 101681274 B
(45)授权公告日 2014.12.17

(21)申请号 200880015942.X
(22)申请日 2008.03.13
(30)优先权数据
11/717,622 2007.03.14 US
(85)PCT国际申请进入国家阶段日
2009.11.13
(86)PCT国际申请的申请数据
PCT/GB2008/000870 2008.03.13
(87)PCT国际申请的公布数据
WO2008/110799 EN 2008.09.18
(73)专利权人 XMOS有限公司
地址 英国布里斯托尔
(72)发明人 戴维·梅
(74)专利代理机构 北京康信知识产权代理有限责任公司 11240
代理人 余刚 吴孟秋
(51)Int.Cl.
G06F 9/48(2006.01)
G06F 1/32(2006.01)
审查员 齐慧峰

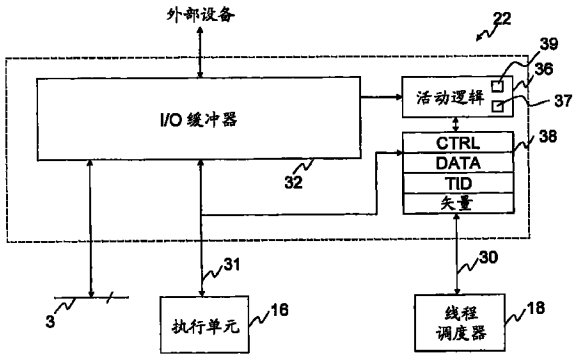
权利要求书5页 说明书19页 附图6页

(54)发明名称

接口处理器

(57)摘要

本发明提供了一种处理器，其包括：第一端口，用于根据第一端口处的第一活动生成第一指示；第二端口，用于根据第二端口处的第二活动生成第二指示。该处理器还包括执行单元，被设置为执行多个线程；以及线程调度器，被连接以接收指示并被设置为根据这些指示调度多个线程用于由执行单元执行。调度包括暂停线程的执行直到收到各指示。第一活动和第二活动中的每个都与各相应线程相关联。



CN 101681274 B

1. 一种处理器，包括：

第一端口，可操作以基于所述第一端口处的第一活动生成第一指示；

第二端口，可操作以基于所述第二端口处的第二活动生成第二指示；

执行单元，被设置为执行多个线程；

各线程寄存器组，表示所述多个线程中的每个线程；以及

线程调度器，被连接至所述第一端口和所述第二端口，以接收所述指示，以及被设置为基于所述指示来调度所述多个线程用于由所述执行单元执行，所述调度包括暂停所述线程的执行直到收到相应指示；

其中，所述第一活动和所述第二活动中的每一个都与各个相应的线程相关联；

其中，所述线程调度器被设置为将第一线程标识符发送到与所述第一端口相关联的标识符寄存器，其中，所述第一线程标识符涉及暂停的等待所述第一活动的第一线程；并且所述第一指示包括所述第一线程标识符从与所述第一端口相关联的标识符寄存器到所述线程调度器的返回。

2. 根据权利要求 1 所述的处理器，其中，所述线程调度器被设置为将第二线程标识符发送到与所述第二端口相关联的标识符寄存器，其中，所述第二线程标识符涉及等待所述第二活动的第二线程；并且所述第二指示包括所述第二线程标识符从与所述第二端口相关联的标识符寄存器到所述线程调度器的返回。

3. 根据权利要求 1 所述的处理器，其中，一旦暂停的线程的所有线程标识符都已经被发送到与所述第一端口相关联的标识符寄存器，则所述处理器适于关闭所述执行单元、所述线程调度器以及时钟中的至少一个。

4. 根据权利要求 3 所述的处理器，其中，一旦所述线程标识符中的至少一个从其相应端口返回，则所述处理器适于对所述执行单元、线程调度器和时钟中的至少一个上电。

5. 根据权利要求 1 所述的处理器，其中，所述第一和第二端口中至少一个包括活动处理逻辑电路并且与至少一个用于存储条件的条件寄存器相关联，所述活动处理逻辑电路被设置为监视所述端口处的活动以及根据满足所述条件的所述活动来生成所述指示。

6. 根据权利要求 5 所述的处理器，其中，所述执行单元被设置为将相应条件发送到每个条件寄存器。

7. 根据权利要求 1 所述的处理器，还包括：程序存储器，通过总线连接到所述执行单元，其中，所述线程调度器通过至少一个与所述总线分离的连接，从所述第一和第二端口中的每个接收所述指示。

8. 根据权利要求 1 所述的处理器，其中，所述线程调度器被设置为一旦暂停所述第一线程就将连续点矢量发送至与所述第一端口相关联的矢量寄存器；以及基于所述第一活动，所述连续点矢量被返回到所述执行单元。

9. 根据权利要求 8 所述的处理器，其中，所述第一线程与来自多个端口的活动相关联，每个所述端口可操作以根据该活动生成相应指示，执行单元适于将多个连续点矢量中的每个发送到所述多个端口中的相应端口，根据相应活动返回相应连续点矢量；并且所述线程调度器适于暂停所述第一线程的执行，直到接收到所述相应指示中的至少一个。

10. 根据权利要求 9 所述的处理器，其中，端口事件启用标记与所述第一端口相关联，并且所述线程调度器适于通过使所述端口事件启用标记有效来启用所述第一活动，从而触

发所述调度，以及通过使所述端口事件启用标记无效来禁止所述第一活动触发所述调度。

11. 根据权利要求 10 所述的处理器，其中，线程事件启用标记与所述第一线程相关联；所述线程调度器适于通过使所述线程事件启用标记有效来根据所述第一活动使所述第一线程能够被调度，以及通过使所述线程事件启用标记无效来根据所述第一活动禁止所述第一线程被调度。

12. 根据权利要求 11 所述的处理器，其中，所述端口事件启用标记和所述线程事件启用标记中的至少一个通过所述第一活动的发生而被自动无效。

13. 根据权利要求 11 所述的处理器，其中，所述线程调度器适于暂停第一线程的执行，其中所述第一线程等待来自第三端口的第三活动的第三指示，同时所述端口事件启用标记和所述线程事件启用标记中的至少一个被无效。

14. 根据权利要求 11 至 13 中的任一项所述的处理器，其中，多个相应端口事件启用标记与关联于所述第一线程的所述多个端口中的每个相关联；且所述线程调度器被设置为使所述线程事件启用标记有效以及随后使所述多个端口事件启用标记有效。

15. 根据权利要求 1 所述的处理器，每个所述寄存器组都被分配用于存储涉及所述多个线程中相应一个线程的信息。

16. 根据权利要求 1 所述的处理器，其中，所述处理器在芯片上实现，并且所述第一和第二端口中的至少一个是用于与所述芯片上的另一个处理器通信的内部端口。

17. 根据权利要求 1 所述的处理器，其中，所述处理器在芯片上实现，并且所述第一和第二端口中至少一个是用于与所述芯片外部的设备通信的外部端口。

18. 根据权利要求 17 所述的处理器，其中至少一个外部端口是引脚端口，其用于检测所述芯片的物理边界的引脚处出现的信号上升沿或下降沿。

19. 根据权利要求 17 所述的处理器，其中，至少一个外部端口是数据端口，所述数据端口用于处理所述端口处的一个或多个位的瞬时逻辑电平。

20. 根据权利要求 1 所述的处理器，其中，所述执行单元适于识别并执行用于暂停和运行线程的专用指令。

21. 根据权利要求 15 所述的处理器，还包括在至少两个所述线程寄存器组之间的至少一个通道，其中，所述线程调度器适于基于所述通道上出现的活动来调度线程。

22. 根据权利要求 1 所述的处理器，其中，所述线程调度器额外地适于基于中断来调度线程。

23. 根据权利要求 22 所述的处理器，其中，基于所述中断而调度的所述线程是所述第一线程，所述中断将暂停的所述第一线程中断。

24. 根据权利要求 1 所述的处理器，其中，如果相关联的活动出现在取决于该活动的线程内的指令的执行时或执行之前，则所述线程调度器适于调度线程而无需暂停。

25. 一种在具有第一端口和第二端口的处理器内调度多个线程用于执行的方法，所述方法包括：

使用用于每个线程的各线程寄存器组，表示所述多个线程中的每个线程；

将各相应线程与所述第一端口处的第一活动和所述第二端口处的第二活动相关联；

根据所述第一活动在所述第一端口处生成第一指示；

根据所述第二活动在所述第二端口处生成第二指示；以及

基于所述指示调度所述线程用于执行，所述调度包括暂停线程的执行直到接收到相应指示；以及

通过根据所述调度执行所述线程而运行所述线程；

其中，所述方法还包括将与暂停的等待所述第一活动的第一线程相关的第一线程标识符发送至与所述第一端口相关联的标识符寄存器；并且所述第一指示包括所述第一线程标识符从与所述第一端口相关联的标识符寄存器到线程调度器的返回。

26. 根据权利要求 25 所述的方法，还包括将与暂停的等待所述第二活动的第二线程相关的第二线程标识符发送至与所述第二端口相关联的标识符寄存器；并且所述第二指示包括所述第二线程标识符从与所述第二端口相关联的标识符寄存器到所述线程调度器的返回。

27. 根据权利要求 26 所述的方法，还包括：

一旦被中止的线程的所有线程标识符都已经发送到与所述第一端口相关联的标识符寄存器，则对以下中的至少一个断电：用于执行所述线程的执行单元，用于调度所述线程的线程调度器，以及时钟。

28. 根据权利要求 27 所述的方法，还包括：

一旦从相应端口返回至少一个所述线程标识符，则对所述执行单元、线程调度器以及时钟中的至少一个上电。

29. 根据权利要求 25 所述的方法，其中，所述第一和第二端口中的至少一个包括活动处理逻辑电路，并且与至少一个用于存储条件的条件寄存器相关联，所述方法还包括使用所述活动处理逻辑电路来监视所述端口处的活动以及根据满足所述条件的所述活动生成所述指示。

30. 根据权利要求 29 所述的方法，还包括将相应条件发送至每个条件寄存器。

31. 根据权利要求 25 所述的方法，其中，所述处理器包括通过总线连接到所述执行单元的存储器，以及所述方法还包括通过至少一条与所述总线分离的连接从所述第一和第二端口中的每一个发送所述指示。

32. 根据权利要求 25 所述的方法，还包括一旦暂停所述第一线程，则将连续点矢量发送到与所述第一端口相关联的矢量寄存器；以及根据所述第一活动返回所述连续点矢量。

33. 根据权利要求 32 所述的方法，包括将所述第一线程与来自多个端口的活动相关联，根据该活动生成相应指示，以及将多个连续点矢量中的每个发送到所述多个端口中的相应一个，根据相应活动返回相应连续点矢量，其中，所述调度包括暂停所述第一线程的执行，直到接收到所述相应指示中的至少一个。

34. 根据权利要求 32 所述的方法，包括通过使端口事件启用标记有效来启用所述第一活动以触发所述调度，以及通过使所述端口事件启用标记无效来禁止所述第一活动触发所述调度。

35. 根据权利要求 34 所述的方法，包括通过使线程事件启用标记有效来根据所述第一活动使所述第一线程能够被调度，以及通过使所述线程事件启用标记无效来根据所述第一活动禁止所述第一线程被调度。

36. 根据权利要求 35 所述的方法，包括通过所述第一活动的发生自动使所述端口事件启用标记和所述线程事件启用标记中的至少一个无效。

37. 根据权利要求 35 所述的方法，还包括：

在使所述端口事件启用标记和所述线程事件启用标记中至少一个无效后，恢复所述第一线程的执行以及随后暂停所述第一线程的执行，该第一线程等待来自第三端口的第三活动的第三指示，同时使所述端口事件启用标记和所述线程事件启用标记中至少一个被无效；

使所述端口事件启用标记和所述线程事件启用标记中的至少一个再次有效；以及

在使所述端口事件启用标记和所述线程事件启用标记中至少一个再次有效之后，暂停正在等待所述第一活动再次发生的所述线程的执行。

38. 根据权利要求 35 至 37 中任一项所述的方法，其中，多个相应端口事件启用标记与关联于所述第一线程的所述多个端口中的每一个相关联；以及所述方法包括使所述线程事件启用标记有效以及随后使所述多个端口事件启用标记有效。

39. 根据权利要求 25 所述的方法，其中，将表示每个所述线程的信息存储在相应线程寄存器组中。

40. 根据权利要求 25 所述的方法，其中，所述处理器在芯片上实现，以及所述第一和第二端口中的至少一个是用于与所述芯片上另一个处理器进行通信的内部端口。

41. 根据权利要求 25 所述的方法，其中，所述处理器在芯片上实现，以及所述第一和第二端口中的至少一个是用于与所述芯片外的设备进行通信的外部端口。

42. 根据权利要求 41 所述的方法，其中，至少一个外部端口是用于检测所述芯片的物理边界的引脚处出现的信号上升沿或信号下降沿的引脚端口。

43. 根据权利要求 41 所述的方法，其中，至少一个外部端口是数据端口，用于处理在所述端口处的一个或多个位的瞬时逻辑电平。

44. 根据权利要求 25 所述的方法，还包括执行用于暂停和运行线程的专用指令。

45. 根据权利要求 39 所述的方法，其中，所述处理器还包括在至少两个所述寄存器组之间的至少一个通道，以及所述方法包括基于所述通道上发生的活动来调度线程。

46. 根据权利要求 25 所述的方法，还包括基于中断来调度线程。

47. 根据权利要求 46 所述的方法，其中，基于所述中断而调度的所述线程是所述第一线程，以及所述中断将被暂停的所述第一线程中断。

48. 根据权利要求 25 所述的方法，如果相关联的活动发生在取决于该活动的线程内的指令执行时或执行之前，则调度线程而无需暂停。

49. 一种移动终端，具有移动应用处理器，至少一个外围设备，以及连接在所述移动应用处理器与所述外围设备之间的接口处理器，所述接口处理器包括：

第一端口，可操作以根据所述第一端口处的第一活动生成第一指示；

第二端口，可操作以根据所述第二端口处的第二活动生成第二指示；

执行单元，被设置为执行多个线程；

各组线程寄存器，表示所述多个线程中的每个线程；以及

线程调度器，被连接至所述第一端口和所述第二端口，以接收所述指示以及被设置为基于所述指示来调度所述多个线程用于由所述执行单元执行，所述调度包括暂停线程的执行直到接收到相应指示；

其中，所述第一活动和所述第二活动中的每个都与各相应线程相关联；

其中，所述线程调度器被设置为将第一线程标识符发送到与所述第一端口相关联的标识符寄存器，其中，所述第一线程标识符涉及暂停的等待所述第一活动的第一线程；并且所述第一指示包括所述第一线程标识符从与所述第一端口相关联的标识符寄存器到所述线程调度器的返回。

50. 一种互连的处理器阵列，至少一个所述处理器包括：

第一端口，可操作以根据所述第一端口处的第一活动生成第一指示；

第二端口，可操作以根据所述第二端口处的第二活动生成第二指示；

执行单元，被设置为执行多个线程；

各组线程寄存器，表示所述多个线程中的每个线程；以及

线程调度器，被连接至所述第一端口和所述第二端口，以接收所述指示，以及被设置为基于所述指示调度所述多个线程用于由所述执行单元执行，所述调度包括暂停线程的执行直到接收到相应指示；

其中，所述第一活动和所述第二活动中的每个都与各相应线程相关联；以及

所述第一和第二端口中的至少一个将所述至少一个处理器与阵列中另一个处理器相连接；以及

其中，所述线程调度器被设置为将第一线程标识符发送到与所述第一端口相关联的标识符寄存器，其中，所述第一线程标识符涉及暂停的等待所述第一活动的第一线程；并且所述第一指示包括所述第一线程标识符从与所述第一端口相关联的标识符寄存器到所述线程调度器的返回。

接口处理器

技术领域

[0001] 本发明涉及接口处理器，该接口处理器是特别用于但不专门用于与其他处理器或外部设备接口的处理器。

背景技术

[0002] 处理器设计者面临的一个挑战是处理数目持续增加的要与处理器通信的外部设备。通常，这是通过为处理器提供某些类型的中断处理能力 (interrupt handling capability) 实现的，该处理器处理连接到外部设备的端口产生的活动。日益增加地，更为复杂性的接口逻辑被用于这些端口来处理，例如，每个端口的多个外部设备。

[0003] 大量不同情形中都需要接口。这里作为背景例子讨论的一个情形是移动应用处理。

[0004] 图 1 示出移动应用处理器 2 的示例性应用。应用处理器 2 包括 CPU 4 和与多个外部设备 8 接口的多个接口控制器 6。接口控制器包括 ：与硬驱 (HDD)8a 和 SDRAM 存储器 8b 接口的存储器控制器 6a ；与摄像机 8c 接口的视频控制器 6b ；与 LCD 显示器 8d 接口的显示器控制器 6c ；与麦克风 8e、扬声器 8f 和耳机 8g 接口的音频控制器 6d ；和与键盘 8h、通用串行总线 (USB) 设备 8i、安全数字 (SD) 卡 8j、多媒体卡 (MMC)8k，以及通用异步接收器 / 发送器 (UART)8i 接口的连接控制器 6e。接口控制器 6 通常经总线 3 连接到 CPU 4。系统还包括功率控制器 10 和无线电处理器 12。

[0005] 注意所示接口控制器 6 多少具有示意性，但通常表示某些类型的专用 I/O 逻辑或特别配置的端口。

[0006] 传统上，外部接口是用中断或利用轮询实现的。在使用中断时，外部外围设备发送信号告知处理器其有数据准备好输入给处理器或向处理器要求数据。在使用轮询时，处理器不断检查设备的状态从而确定是否设备准备好提供或接受数据。

[0007] 如图 1 所示的应用程序处理器 2 的一个可能的实施是使用专用集成电路微控制器 (ASIC)。ASIC 是硬连线的设备，其可能包括专用于特定应用以及经优化适于该应用的微处理器。对于给定的功能，与其他选择相比，它们通常更便宜并且能耗更低。然而，它们计复杂，必须预设计且不能容易地重配置。

[0008] 另一种可能是使用现场可编程门阵列 (FPGA) 设备。FPGA 是半导体设备，其可在制造后现场配置。为了配置 FPGA，首先利用计算机建模所需的逻辑功能，例如通过绘制示意图或创建描述功能的文本文件。FPGA 包括经静态配置的互连线进行通信的查询表阵列。计算机模型是用 FPGA 卖家提供的软件编译的，这产生能够下载到 FPGA 查询表的二进制文件。这允许装备制造商设计 FPGA 从而满足其自身的个性化需要。

[0009] 在该例子中，接口控制器 6 作为 FPGA 实施。这具有这样的益处，即移动电话制造商可购买通用 FPGA 设备 2，然后在当场 (即“现场”) 将其配置成专用于他们所需要应用的。然而，FPGA 的缺点是与 ASIC 相比更为昂贵，速度更缓慢以及能耗较高。

[0010] 在可选的例子中，整个芯片 2 可以用 FPGA 实现，或芯片 2 可以是具有连接在芯片

2 和各外围设备 8 之间的分离的 FPGA 芯片的通用目的处理器。然而，这些选择可能更为昂贵以及功耗更高——限制了多数移动电话和其他消费电子设备。

[0011] 以 ASIC 的价格、速度、范围和能耗水平实现 FPGA 的可配置性会是有利的。

发明内容

[0012] 按照本发明的一个方面，提供了一种处理器，其包括：第一端口，该第一端口用于基于第一端口处的第一活动生成第一指示；第二端口，该第二端口用于基于第二端口处的第二活动生成第二指示；执行单元，其被设置为执行多个线程；线程调度器 (threadscheduler)，其被连接以接收所述指示以及被设置为调度所述多个线程用于执行单元基于所述指示而执行，所述调度包括暂停 (suspending) 线程的执行直到接收到相应指示；其中第一活动和第二活动分别与各相应线程关联。

[0013] 因为每个线程与各端口处的活动相关联，以及线程调度器被连接以暂停该活动的线程等待 (pending) 指示，则处理器有利地“预先准备好”以快速响应发生在端口的活动。因此本发明提供了在需要快速响应外部激励的应用中的显著改进。另外，在线程被暂停时，如果仍然要调度其他线程，则线程调度器可开始调度这些线程。

[0014] 在要求快速反应时间时，暂停的线程方法对使用中断是优选的。使用暂停的线程，线程调度器可准备执行预期关联活动的线程，如上面讨论的那样，这意味着处理器做好了反应的准备。相反，使用中断，执行单元由来自外部设备的信号中断，同时执行某些潜在无关代码。在中断情形中，当前程序状态必须在中断起作用前被保存。因此，使用中断的反应时间非常慢。轮询的能量效率较低，因为要求不断的询问和应答。

[0015] 线程调度器可设置为将与暂停等待第一事件的第一线程相关的第一线程标识符传输到与第一端口关联的标识符寄存器。线程调度器可设置为将与暂停等待第二事件的第二线程相关的第二线程标识符传输到与第二端口关联的标识符寄存器。在这样的实施例中，线程调度器适于管理线程的执行，但在其遇到取决于特定活动的线程内指令时，线程调度器可有利地通过将其责任传给各与其关联的端口而“撇开 (set aside)”线程。这也有助于反应时间。

[0016] 线程调度器可设置为在暂停所述第一线程后将连续点矢量 (continuation point vector) 传送至与第一端口关联的矢量寄存器；且连续点矢量可根据第一活动返回给执行单元。连续点矢量标识代码中某个点，在各活动发生时执行应在该点继续。这可有利地允许线程额外的责任 (responsibility) 发送到端口，以及可释放不必以等待 (pending) 指令填充的指令缓冲器。

[0017] 另外，使用连续点矢量可允许每个线程处理多个活动（或事件）。因此，第一线程可与来自多个端口的活动关联，每个端口用于生成依赖于该活动的各指示；执行单元可适于传输多个连续点矢量的每个至各所述多个端口之一，各连续点矢量是根据各活动返回的，且线程调度器可适于暂停第一线程的执行，直到接收各指示中的至少一个。

[0018] 一旦暂停线程的所有线程标识符传输到端口，处理器可以关闭 (power down) 执行单元、线程调度器和时钟中的至少一个。当从各处理器端口返回至少一个线程标识符时，处理器可以上电 (powerup) 所述执行单元、线程调度器和时钟中的至少一个。因此，如果所有线程已经调度以及按各端口活动交给各端口，然后处理器可以关闭部分或全部组件直到发

生一个活动。这显著改善系统的功耗。

[0019] 第一和第二端口中至少一个可包括活动处理逻辑以及可与至少一个用于存储条件的条件寄存器(condition register)关联,活动处理逻辑被设置为用于监视端口的活动以及根据满足所述条件的所述活动生成所述指示。执行单元可设置为传送各条件至每个条件寄存器。有利地,这通过允许线程调度器为端口提供条件改善了系统的通用性,以使端口可处理更特殊的和多样的条件类型。

[0020] 端口事件启用标记可与第一端口关联,且线程调度器可适于通过使端口事件启用标记有效(assert)使得所述第一活动触发所述调度,以及通过使端口事件启用标记无效(de-assert)而禁止第一活动触发所述调度。线程事件启用标记可与第一线程关联;而线程调度器可适于通过使线程事件启用标记有效,根据第一活动使第一线程能够被调度,以及通过使线程事件启用标记无效,根据第一活动禁止调度第一线程。线程调度器可适于暂停第一线程的执行,该第一线程等待来自端口的第三活动的第三指示,同时端口事件启用标记和线程事件启用标记中至少一个被无效。

[0021] 端口事件启用标记和线程事件启用标记中至少一个可被第一活动的生成自动无效。

[0022] 多个各自端口事件启用标记可与多个与第一线程关联的端口中的每个关联;且线程调度器可设置为使线程启用标记有效,以及随后使所述多个端口事件启用标记有效。

[0023] 处理器可包括由总线连接到执行单元的程序存储器,其中线程调度器通过至少一个与所述总线分离的连接而连接以接收来自第一和第二端口中每个端口的所述指示。这消除了端口和线程调度器间的通信中总线仲裁的需求,因此进一步改善处理器的反应时间。

[0024] 处理器可包括多个线程寄存器组,每组线程寄存器都被分配来存储关于所述多个线程的各个线程的信息。处理器可包括在至少两组所述线程寄存器间的至少一个通道,且线程调度器可适于根据在所述通道上发生的活动调度线程。

[0025] 处理器可在芯片上实施。所述第一和第二端口中的至少一个可以是与所述芯片上另一个处理器通信的内部端口。所述第一和第二端口中的至少一个可以是与所述芯片外部设备通信的外部端口。至少一个外部端口可以是用于检测出现在芯片物理边界处引脚(pin)上的信号的上升沿或下降沿的引脚端口。至少一个外部端口可以是用于处理端口处一个或多个位的瞬时逻辑电平(logic-level)的数据端口。

[0026] 执行单元可适于识别和执行用于暂停和运行线程的专用指令。

[0027] 线程调度器还可以适于基于中断对线程进行调度。基于中断被调度的线程是第一线程,当暂停时该中断将第一线程中断。

[0028] 如果关联活动发生在取决于该活动的线程内的指令执行时或执行之前,则线程调度器可适于调度线程而无需暂停。

[0029] 根据本发明的进一步方面,提供了调度多个线程的方法,用于具有第一和第二端口的处理器中的执行(execution),该方法包括:将第一端口的第一活动及第二端口的第二活动与各相应线程相关联;根据第一活动在第一端口生成第一指示;根据第二活动在第二端口生成第二指示;基于所述指示调度线程用于执行,所述调度包括暂停线程的执行直到接收各指示;以及根据所述调度通过执行线程而运行线程。

[0030] 根据本发明的另一个方面,提供了端口,用于调度多个线程以便执行,该端口包

括：活动处理逻辑电路，设置为根据端口处的活动生成指示，寄存器，存储用于识别与所述活动关联的线程的信息，以及寄存器，用于存储一个条件；其中所述活动处理逻辑电路适于在所述活动满足所述条件时传输芯片上第一指示。

[0031] 根据本发明的另一个方面，提供了线程调度器，其用于调度具有第一端口和第二端口的处理器内多个线程以用于执行，其中：该线程调度器适于接收第一端口根据第一端口的第一活动生成的第一指示；该线程调度器适于接收第二端口根据第二端口的第二活动生成的第二指示；且线程调度器适于基于所述指示调度所述多个线程用于通过该单元执行来执行，所述调度包括暂停线程的执行直到接收各指示；其中第一活动和第二活动中的每一个都与各相应线程关联。

[0032] 按照本发明的另一个方面，提供了具有移动应用程序处理器的移动终端，至少一个外围设备，和连接在移动应用程序处理器与外围设备之间的接口处理器，接口处理器包括：第一端口，根据第一端口处的第一活动用于生成第一指示；第二端口，根据第二端口处的第二活动生成第二指示；执行单元，设置为执行多个线程；和线程调度器，被连接以接收所述指示以及设置为根据所述指示调度所述多个线程用于该执行单元执行，所述调度包括暂停线程的执行直到接收各指示；其中第一活动和第二活动中的每一个都与各相应线程关联。

[0033] 按照本发明的另一个方面，提供了互连处理器阵列，至少一个所述处理器包括：用于根据第一端口处的第一活动生成第一指示的第一端口；用于根据第二端口处的第二活动生成第二指示的第二端口；设置为执行多个线程的执行单元；以及线程调度器，被连接以接收所述指示以及被设置为基于所述指示来调度所述多个线程用于该执行单元执行；所述调度包括暂停线程的执行直到接收到各个指示，其中第一活动和第二活动每个都与各相应线程关联；以及所述第一和第二端口中的至少一个连接阵列中至少一个处理器和另一个处理器。

[0034] 为了更好地理解本发明以及示出其如何实施，作为例子要参考相应附图。

附图说明

[0035] 图 1 示出本领域已知的应用 FPGA 设备的例子；

[0036] 图 2 示出接口处理器的应用例子；

[0037] 图 2a 示出接口处理器的另一个应用例子；

[0038] 图 3 是接口处理器架构的示意表示；

[0039] 图 4 是端口的示意表示；

[0040] 图 5 是线程寄存器组的示意表示；

[0041] 图 6 是示出线程调度器操作的流程图；

[0042] 图 7 是线程寄存器组间通道的示意图；以及

[0043] 图 7A 是通道末端 (channel end) 的示意图。

具体实施方式

[0044] 图 2 示出移动电话中接口处理器的示例性应用。移动应用处理器 2 需要与多个外围设备 8 通信。如图 1 所示，应用处理器 2 包括总线 3，CPU 4，以及与硬驱 (HDD)8a 和 SDRAM

存储器 8b 接口的存储器控制器 6a，以及功率控制器 10 和无线电处理器 12。

[0045] 然而，取代专用控制器 6，图 2 的装置允许 CPU 4 经通用端口 7 与外部通信。在该例子中，提供了通用端口 7a 和 7b 用于与摄像机 8c 和 LCD 显示器 8d 接口；提供了通用端口 7c 用于与麦克风 8e、扬声器 8f 和耳机 8g 接口；提供了通用端口 7d 用于与键盘 8h、通用串行总线 (USB) 设备 8i、安全数字 (SD) 卡 8j、多媒体卡 (MMC)8k，以及通用异步接收器 / 发射器 (UART) 设备 8l 接口。

[0046] 在图 2 中，接口处理器 14a、14b 和 14c 设置在相关端口 7 的输出处，其中第一接口处理器 14a 连接在图像设备 8c-8d 和通用端口 7a-7b 之间，第二接口处理器 14b 连接在视频设备 8e-8g 间，以及第三接口处理器 14c 连接在通用端口 7d 和不同连接设备 8h-8m 间。端口 7 仅需要是一般用途端口，因为专用显示器，音频和连接功能是由接口处理器 14a-14c 以后面详述的方式实现的。端口 7 不需要使用 FPGA 逻辑电路，因为接口处理器 14 提供灵活性和可配置性，否则要由 FPGA 提供。接口处理器 14a 具有连接到端口 7a 和 7b 的端口 22a 和 22b，连接到外部设备 8c 和 8g 的端口 22c，22d，22e 和 22f。接口处理器 14b 和 14c 具有类似端口，图 2 中没有示出。

[0047] 接口处理器通常涉及执行用来经接口发送数据的特定协议，再格式化数据包括将数据在并行格式 (parallel format) 和串行格式 (serial format) 间进行转换，以及可能的更高水平功能，如对其编码、压缩或加密。

[0048] 接口处理器的另一个应用是在图 2a 中所示的多处理器芯片 202 中作为核 (tile)。这样的芯片 202 使用高性能互连 204，该高性能互连 204 支持芯片 202 上处理器 14 和芯片间链路 206 之间的通信以便系统可容易地由多个芯片构造。每个处理器 14 经端口 22 连接到其芯片间链路 206 和高性能互连线 204。

[0049] 下面更充分讨论的接口处理器的重要特征是其管理端口 22 处的活动的能力。每个接口处理器包括 CPU、存储器和通信器。为了允许 CPU 与端口间的直接连接和响应性连接，每个处理器具有用于执行大量同时程序线程 (concurrent program threads) 的硬件支持，每个都包括指令序列，以及至少其中一些专门负责处理端口的活动。如下面更详细的讨论，硬件支持包括：

[0050] - 用于每个线程的寄存器组，

[0051] - 动态选择执行哪个线程的线程调度器，

[0052] - 用于输入和输出的端口组 (端口 22)，

[0053] 每个处理器上使用一小组线程可用来允许通信或输入 / 输出以与处理器处理的其他等待任务一起进行，以及通过当其他线程暂停等待与远程接口处理器间通信时允许某些线程继续而允许互连中等待隐藏 (latency hiding)。

[0054] 图 3 示意地示出根据本发明一个实施例的接口处理器 14 的示例性架构。处理器 14 包括在线程调度器 18 控制下执行指令线程的执行单元 16。处理器 14 还包括保存程序代码和其他数据的随机存取存储器 (RAM)24，以及用于存储诸如起动代码 (boot code) 的永久信息的只读存储器 (ROM)(未示出)。

[0055] 线程调度器 18 动态选择哪个线程应由执行单元 16 执行。传统上，线程调度器的功能将仅从程序存储器中调度线程，以便保持处理器被完全占用。然而，按照本发明，线程调度器 18 的调度也涉及端口 22 处的活动。这方面应该注意的是，线程调度器可直接耦合

到端口 22 以便在线程由于端口处的输入或输出活动而变得可运行时最小化延迟。

[0056] 线程调度器 18 考虑的 m 个线程中的每个线程由线程调度器 18 存取的寄存器库 (bank)20 中各组线程寄存器 $20_1 \ldots 20_m$ 表示。还被提供了指令缓冲器 (INSTR)19 用于在将从存储器 24 取出的指令传入执行单元 16 之前临时保存该指令。这些寄存器和缓冲器的细节在下面讨论。

[0057] m 个线程中，线程调度器 18 保持一组 n 个可运行线程，该组线程被称为“运行”，线程调度器从该组中优选地以循环方式 (round-robin manner) 依次提取指令。当线程不能继续时，则通过将其从运行组中移除而使其暂停。其原因可以是例如，因为线程正等待一个或多个以下类型的活动：

[0058] - 在线程能够运行之前其寄存器被初始化，

[0059] - 线程尝试从未就绪或没有可用数据的端口输入，

[0060] - 线程尝试对未就绪或没有用于数据的空间的端口输出，

[0061] - 线程执行了引起线程等待一个或多个事件的指令，该事件可以是端口准备好输入时生成的。

[0062] 注意这里所用的术语“事件”是指特殊类型的操作，其与基本输入 - 输出操作稍有不同。下面结合图 4 和 5 讨论所述的区别。

[0063] 有利地，为了有助于快速反应时间，在线程调度器 18 和执行单元 16 之间提供直接硬线连接 28，以允许线程调度器 18 控制执行单元 16 应取出并执行哪个或哪些线程。直接硬线路径 30a，30b，30c 也提供在线程调度器 18 和每个端口 22 之间；以及在线程调度器 18 和每个寄存器 20 之间提供直接硬线路径 $29_1 \ldots 29_m$。这些直接路径优选地提供控制路径，其允许线程调度器将各线程与一个或多个端口 22 相关联，以及特别地当某些活动发生时从端口返回就绪指示，这样允许处理器快速响应发生在端口 22 的活动或激励。线程调度器涉及端口的操作将在下面结合图 4 和 6 讨论。

[0064] 执行单元 16 还通过直接连接 27 和 31 访问每个端口 22a-22c 以及每个寄存器 20_1-20_m，因此在芯处理器、寄存器、和外部环境间提供了直接链接。优选地，这些直接路径提供进一步的控制路径以允许执行单元将条件发送给端口。这将在下面结合图 4 进一步详细讨论。直接路径 27 和 31 也可允许数据在线程调度器 20 和端口 22 间直接输入和输出，因此允许线程直接与外部环境通信。例如，数据可直接从外部设备写入到线程的操作数寄存器之一，而非写入到存储器 24 以及随后从其中取出。相反，在一次运算之后，来自操作数寄存器的数据可由执行单元 16 拾取并直接发送出端口 22。这显著改善了反应时间。

[0065] 值得注意的是“直接连接”或“直接路径”表示与执行单元和程序存储器 24 之间的连接相分离的连接。因此，例如线程调度器 18 和执行单元 16 可存取从端口 22 输入的数据，而不必将数据存储在存储器 24 中以及随后从其中取出。特别地，如果执行单元 16 和存储器 24 之间是通过总线 3 连接的，则“直接”连接或路径意味着与总线相分离的连接或路径。因此，端口 22、寄存器 20、线程调度器 18 和执行单元 16 间的各种通信都可发生而无需总线仲裁，改善反应时间。端口 22 还可提供与总线 13 的额外连接 (未示出)。

[0066] 本申请中所用的术语“端口”可指“针式端口 (pin port)”或“数据端口”。针式端口负责检测单独的逻辑变换，即发生在处理器芯片物理边界处引脚上的信号的上升沿和下降沿。数据端口是“更高层次 (higher level)”，因为它们可处理一个或多个位，通常累

加在 I/O 缓冲器中，以及通常构成数据的一部分，如字。不用检测上升沿和下降沿，数据端口处理特定瞬间的一个或多个位的状态或逻辑电平。数据端口可以是开 / 关芯片 (on/off chip)，或可以是对嵌在同一芯片上的另一处理器的端口。注意“针式端口”和“数据端口”实际上可指同一实际端口的不同模式。

[0067] 图 4 示意地示出按照本发明优选实施例的端口 22。端口 22 包括 I/O 缓冲器 32 用于发送来自或者送往处理器 14 的数据。此外，每个端口 22 包括活动操纵逻辑 36 用于监视发生在端口的活动以及利用至少一个就绪数位或标记 37 将特定活动的发生信号化。就绪标记 37 优选地通过直接路径 30 发信号告知线程调度器。端口可检测的潜在活动包括 ：

[0068] - 数据已经输入到端口

[0069] - 某些特定数据已经输入到端口，以及 / 或

[0070] - 端口已经可用于输出。

[0071] 为了有助于对这些活动的检测，端口 22 被提供了一组寄存器 38。这些包括线程标识符 (TID) 寄存器用于存储相关线程的标识 (identification)，控制 (CTRL) 寄存器用于存储一个或多个条件，连续点矢量 (VECTOR) 寄存器用于存储程序中执行被暂停的位置，以及数据 (DATA) 寄存器用于存储任何与条件关联的数据。值 TID 由线程调度器 18 通过直接路径 30(图 3 中直接路径是 30a，30b，30c) 写入到寄存器 38 中，以及值 VECTOR、CTR 和 DATA 是由执行单元 16 通过直接路径 31 写入的。一旦检测到所需活动，TID 返回到线程调度器 18，以便识别关联的线程。活动逻辑还包括下面进一步详细讨论的启用标记 39。

[0072] 注意，虽然图 4 中所示寄存器 38 包含在端口 22 内，它们实际上可位于处理器 14 中任何其他地方并仅与端口 22 关联。

[0073] 图 5 示出用来表示线程的示例性线程寄存器库 20。寄存器库 20 包括对应于当前由线程调度器 18 所考虑的各线程 T_1 到 T_m 的多个寄存器组。在该优选例子中，每个线程的状态由 18 个寄存器表示 ：2 个控制寄存器，4 个存取寄存器和 12 个操作数寄存器。这些寄存器分别如下 ：

[0074] 控制寄存器 ：

[0075] -PC 是程序计数器

[0076] -SR 是状态寄存器

[0077] 存取寄存器 ：

[0078] -GP 是全局公用指针 (global pool pointer)

[0079] -DP 是数据指针

[0080] -SP 是栈指针

[0081] -LR 是链接寄存器

[0082] 操作数寄存器 :OP1...OP12

[0083] 控制寄存器存储关于线程状态以及用于控制线程执行的信息。特别地，线程接受事件或中断的能力由保存在线程状态寄存器 SR 中的信息控制。存取寄存器包括用于进程的局部变量的栈指针，通常用于进程间共享数据的数据指针，以及用于存取大常数和进程入口点 (procedure entry points) 的常数公用指针 (constant poolpointer)。操作数寄存器 OP1...OP12 由指令使用，该指令执行算术和逻辑操作，存取数据结构，以及调用子程序。

[0084] 还提供了大量指令缓冲器 (INSTR)19 用于临时存储线程实际指令。每个指令缓冲器优选为 64 位长，且每个指令优选为 16 位长，允许每个缓冲器存储 4 个指令。指令是在线程调度器 18 的控制下从程序存储器 24 取出的并被临时存放在指令缓冲器 19 中。

[0085] 执行单元可存取每个寄存器 20 和缓冲器 19。进一步，线程调度器 18 至少可为每个线程存取状态寄存器 SR。

[0086] 如上所述，这里所用的术语“事件”指特定类型的操作，或对应于特定类型操作的活动。基于操作的事件与基本输入 - 输出操作稍有不同，且以如下方式工作。事件首先是通过从执行单元 16 将连续点矢量和从线程调度器 18 将线程标识符转移到与端口 22 关联的 VECTOR 和 TID 寄存器 38 而为线程设定的，优选地经直接路径 31 和 30。关联的条件和条件数据还可写入端口 22 的 CTRL 和 DATA 寄存器 38 中。这样在端口设定事件，但不必启用。为了使端口生成事件指示，还必须 (优选地由线程调度器 18 通过直接路径 30) 使端口的启用标记 39 有效。此外，为了使线程自身接受事件，线程的各状态寄存器 SR 中的线程事件启用 (EE) 标记必须被设定为事件启用的。一旦事件这样设定并启用，线程就可以使用基于事件 (event-based) 的等待指令暂停等待事件，该基于事件的等待指令作用于线程调度器 18。此时，当前等待指令可从相关指令缓冲器 19 中丢弃。当事件发生，如某些数据输入到端口时，通过从端口 22 返回到线程调度器 18 和执行单元 16 的线程标识符和连续点矢量，信号告知该发生，允许连续点矢量处的指令从程序存储器 24 取出以及存入指令缓冲器 19，以及执行在代码中适当位置点继续。

[0087] 当事件发生时，各状态寄存器 SR 中线程 EE 标记可设定为事件禁止以防止线程在事件发生后立即对事件做出反应。作为事件发生时线程执行指令的结果，启用标记 39 可被无效。

[0088] 在准备等待来自一个或多个端口的事件中设置多个端口时，可以使启用标记 39 有效。线程 EE 标记还可在启用一组端口启用标记前设定为事件启用 (event-enable) 的，以及在该情形中就绪的即将启用的第一端口将生成事件，该事件通过在连续点矢量立即取出以及执行指令使得当前指令被丢弃以及继续执行。

[0089] 端口启用标记 39 和状态寄存器 EE 标记的优点在于 :事件的启用和禁止是与事件设立和通过等待指令暂停线程分离的，允许不同输入和输出条件准备好用于特定线程和 / 或为各种不同线程开启和关闭 (toggled on and off)。例如，事件可以被留作在端口 22 建立，即使事件被禁止。因此，事件可以由线程重复使用，因为虽然事件已经发生过一次，但是线程标识符，连续点矢量和条件仍然存储在端口 22 的 TID,VECTOR,CTRL 和 DATA 寄存器 38 中。因此，如果线程需要重复使用事件，端口寄存器 38 不需要重复写入，相反，端口的启用标记 39 能够简单地被再次有效，以及 / 或者线程的状态寄存器 SR 中 EE 标记可再次设置为事件启用的。另一个等待指令将暂停等待同一事件再次发生的线程。

[0090] 而且，使用连续点矢量允许每个线程启用多个事件。也就是说，给定的线程可通过将连续点矢量发送到一个端口 22a 而在该端口建立一个事件，通过将不同的连续点矢量发送到另一个端口 22b 而在该端口建立另一个事件，如此类推。线程也可通过分别为各端口使不同的启用标记 39 有效或者无效来逐个启用或禁止各个事件。等待指令将使线程暂停等待任何启用的事件。

[0091] 与事件相比，使用基本 I/O 操作，线程调度器 18 不传输连续点矢量给 VECTOR 寄存

器，且不使用端口的启用标记 39 或状态寄存器 SR 中的 EE 标记。相反，等待指令仅留在指令缓冲器 19 中，且如果需要，根据就绪标记 37 的指示，仅将执行中止以等待输入或端口输出的可用性。在实施例中，仅要求 TID 寄存器按照基本 I/O 进行调度。基本 I/O 可以使用或不使用 CTRL 和 DATA 寄存器中的条件。如果不使用这样的条件，端口一准备好 I/O 就会完成。

[0092] 还须注意的是，一旦在事件之后恢复执行线程，则当然随后可执行基本 I/O 操作。相反，一旦线程在基本 I/O 后继续，则其随后可包括事件操作。任何这类事件和 I/O 都可包括在线程中。例如，基本 I/O 操作可在两个基于事件的等待操作间交叉进行，同时事件被禁止（即，端口启用标记 39 和 / 或状态寄存器的 EE 标记被无效），但事件矢量和条件仍然留作在寄存器 38 中设置。也就是说，事件可在完成第一个基于事件的等待操作后禁止，基本 I/O 随后用同一端口执行，以及然后同一事件被再次启用用于在第二个基于事件的等待操作。如上所述，基本 I/O 操作中止和解除中止该线程，但不影响端口的启用标记 39 或状态寄存器中的 EE 标记，也不发送控制到事件矢量。

[0093] 下面参考图 6 的流程图说明线程调度器和两个示例性端口的操作。在步骤 100，执行单元 16 在线程调度器 18 的指导下开始执行第一和第二线程。在步骤 102，线程调度器遇到第一线程内以事件为条件的部分代码，例如接收端口 22a 处的某些特殊信号。在步骤 104，线程调度器发送第一线程的线程标识符 (TID) 和连续点矢量 (VECTOR) 到端口 22a，该连续点矢量指定一旦检测到事件则线程的执行应该在程序中的哪个点继续，以及任何需要的条件控制信息 (CTRL) 和关联的数据 (DATA)。例如，数据可以是指令预期在端口接收的信号的值。在步骤 104，线程调度器还可设置第一端口的启用标记 39 以及将第一线程的状态寄存器 SR 设置为事件启用的。

[0094] 在步骤 106，端口 22a 从线程调度器 18 接收该信息。在步骤 108，线程调度器 18 暂停第一线程的执行。在步骤 110，端口 22a 开始监视端口处的活动。

[0095] 在步骤 112，线程调度器 18 确定第二线程仍未完成 (outstanding) 且执行单元 16 继续在线程调度器 18 的指导下执行第二线程。在步骤 114，线程调度器 18 遇到以事件为条件的部分代码。在步骤 116，线程调度器 18 发送线程标识符和连续点矢量以及任何其他要求的条件信息至端口 22b。在步骤 116，线程调度器还可设置第二端口的启用标记 39 以及将第二线程的第二状态寄存器设置为事件启用的。在步骤 118，端口 22b 接收该信息。在步骤 120，线程调度器暂停第二线程的执行。在步骤 122，端口 22b 开始监视发生在端口的活动。

[0096] 在步骤 124，线程调度器确定当前不再有需要调度的未完成的线程，以及系统关闭除端口 22a 和 22b 以外的所有组件。在步骤 128，端口 22a 检测到相关事件，例如收到存储在 DATA 寄存器中的信号，以及因此返回线程标识符 (TID) 以及续点矢量 (VECTOR)（以及将第一线程的状态寄存器设置为事件禁止）。在步骤 126，线程调度器 18 接收返回的标识符。现在执行可以继续，那么在步骤 130，系统再次上电。在步骤 134，执行单元 16 在线程调度器 18 的指导下完成第一线程的执行。在步骤 138，端口 22b 为第二线程检测相关事件以及返回其线程标识符以及连续点矢量（以及将第二线程的状态寄存器设置为事件禁止）。在步骤 136，线程调度器 18 接收返回的信息，在步骤 138，执行单元 16 在线程调度器 18 的控制下完成第二线程的执行。注意在步骤 134 和 136 间可以有额外的断电步骤。

[0097] 如图 7 所示，本发明的原理也可扩展到线程间的通信，或更精确地，线程寄存器组 20（其存储表示线程的信息）间的通信。为了说明的目的，仅 4 个线程寄存器组 20_1 到 20_4 在图 7 中示出，每个存储各线程 T_1 到 T_4 的信息。每个线程寄存器组都通过互连系统 40 连接到各其他组，互连系统用于建立至少一个通道用于直接在至少两个线程寄存器组 20 间发送数据。通道优选地用来与操作数寄存器 OP 交换数据，但原则上用来与（诸如状态寄存器 SR 的）其他类型的寄存器交换信息。线程调度器 18 可基于发生在通道上的活动以如上述的与端口相关的类似方式来调度线程。

[0098] 如图 7A 所示，每个通道末端 42 像一对端口，具有输入缓冲器 44 和输出缓冲器 46。类似于端口 22，每个通道输入缓冲器 44 和输出缓冲器 46 可具有活动处理逻辑 36’用于监视发生在通道上的活动以及通过至少一个就绪标记 37’来发信号告知某些活动的发生。通道末端可检测的潜在活动包括：数据已经输入到通道，或通道已经可用于输出。如果在通道太满而不能提取数据时执行输出指令，则执行指令的线程被中止，以及在通道中有足够空间用于指令成功完成时再开始。类似地，当指令被执行且没有足够的数据可用时，线程被中止以及当有足够数据可用时再开始。

[0099] 再如端口 22 一样，为了有助于对这样的活动的检测，每个通道末端与寄存器 38’关联。这些寄存器包括用于存储相关线程标识的线程标识符 (TID) 寄存器，以及用于存储根据事件的发生执行应该在程序的哪个位置继续的连续点矢量 (VECTOR) 寄存器。这些 TID 和 VECTOR 寄存器可由线程调度器 18 以及执行单元 16 使用从而以与端口 22 相同的方式调度线程。VECTOR 寄存器允许通道生成事件以及中断。通道末端还具有启用标记 39’用于启用通道以生成事件。在实施例中，通道末端 42 可以不具有 CTRL 和 DATA 寄存器。

[0100] 同一通道末端 42 还可以用来将数据从线程寄存器经端口 22 传送到外部环境。也就是说，执行单元 16 可经通道末端 42 拾取寄存器 20 的内容并经端口 22 将其直接传出；相反，执行单元 16 还可以从端口 22 接收输入以及经通道末端 42 将其直接发送给寄存器 20。因此，如果按照本发明的两个或多个接口处理器连接到一起，例如图 2A 中所示，通道也可在这些处理器间建立。在处理器间系统内，每个处理器可以通过许多双向物理链路 204 与系统互连相接口，这允许与其他处理器同时连接。

[0101] 这里使用的包括端口、通道以及活动的其他源的总术语是“资源”。

[0102] 接口处理器可支持几种编程方法，这是由于其基于线程的结构。接口处理器可以作为执行标准输入和输出的单个常规处理器，或可编程为数百个通信组件并行阵列的一部分。本发明提供了支持这些可选做法的指令集合。指令集合包括支持初始化、终止、开始和停止线程以及提供输入 / 输出通信的特殊指令。输入和输出指令允许与外部设备非常快的通信。它们支持高速，低时延输入和输出以及高级协同编程技术。这里它们用于处理端口活动，这将在下面更充分地讨论，其中说明了用于实施本发明的示例指令。

[0103] 首先为使用 GETR 指令的线程预留资源，GETR 指令指定了所要求的资源类型，并可用 FREER 指令再次释放。

[0104] 端口可用于输入或输出模式。在输入模式中，条件可用来过滤发送到线程的数据。如下面所述，在数据可用时，端口可用来生成事件或中断。这允许线程监视几个端口，仅为就绪的端口服务。然后一旦端口就绪，输入和输出指令，IN 和 OUT 可用来发送数据进出端口。在该情形中，IN 指令输入以及零扩展 (zero-extends)n 位端口中 n 个最不重要的位，

OUT 指令输出该 n 个最不重要的位。

[0105] 两个另外的指令，INSHR 和 OUTSHR 优化数据传输。INSHR 指令将寄存器内容向右移 n 位，用从 n 位端口输入的数据填补最左边 n 位。OUTSHR 指令输出数据的 n 个最不重要的位至 n 位端口以及将寄存器内容向右移 n 位。

[0106] OUTSHR port，s port ◁ s[bits 0 for width(port)]；从其输出

[0107] s ← s ＞＞ width(port) 端口和移位

[0108] INSHR port，s s ← s ＞＞ width(d)； 移位和输入

[0109] port ▷ s[bits(bitsperword-width(d))for width(d)] 来自端口

[0110] 其中“▷”表示输入，“◁”表示输出。

[0111] 端口在可使用前必须被配置。端口是用 SETC 指令配置的，SETC 指令被用来定义端口的几种独立设置。每种设置都具有缺省模式，且仅在需要不同模式时要配置。

[0112] SETC port，mode port[ctrl] ← mode 设置端口控制

[0113] 下面说明 SETC 模式设置的效果。每种设置的首项是缺省模式。

[0114] 模式 效果

[0115] OFF 端口不有效，针脚高阻抗

[0116] ON 有效

[0117] IN 端口是输入

[0118] OUT 端口是输出（但输入返回当前针脚值 (pin value)）

[0119] EVENT 端口将引起事件

[0120] INTERRUPT 端口将唤醒 (raise) 中断

[0121] DRIVE 针脚被高电平和低电平驱动

[0122] PULLDOWN 针脚下拉电平用于 0 位，否则为高阻抗

[0123] PULLUP 针脚上拉电平为用于 1 位，否则为高阻抗

[0124] UNCOND 端口始终就绪，输入立即完成

[0125] EQUAL 当端口值等于其 DATA 值时端口就绪

[0126] NE 当端口值不同于其 DATA 值时端口就绪

[0127] TRANSITION 当端口值向其 DATA 值改变时端口就绪

[0128] GR 当端口值大于其 DATA 值时端口就绪

[0129] LS 当端口值小于其 DATA 值时端口就绪

[0130] DRIVE，PULLDOWN 和 PULLUP 模式仅在端口方向为 OUT 时相关。TRANSITION 条件仅对 1 位端口相关，以及 GR 和 LS 条件仅对 1 位以上的端口相关。

[0131] 每个端口具有就绪位 37，其用来控制通过端口的数据流，以及定义端口是否能够完成输入或输出指令。根据端口配置以不同方式设置就绪位。在 SETC，SETD 或 SETV 中任意指令被执行时，就绪位被清除。

[0132] 输入模式中端口可配置为执行条件输入。条件过滤输入数据，以便仅使符合条件的数据返回给程序。当条件被设定时，IN 和 INSHR 指令将仅在端口就绪时完成。如上所述，在未就绪端口上执行输入指令将中止线程。在就绪时，端口设定其就绪位，这通过信号告知线程调度器。线程继续以及再执行输入指令。这次端口是就绪的，数据被返回以及就绪位 37 被清除。

[0133] 一旦设定端口就绪位，满足条件的数据值被捕获，因此软件获得满足条件的值，即使端口上的值随后改变。当 IN 或 INSHR 指令被执行以及就绪位被设定时，则返回数据以及清除就绪位。如果就绪位没有被设定，则线程被中止直到就绪位被设定。如果条件被设定，则数据与条件比较，以及就绪位仅在满足条件时设定。

[0134] 在执行 OUT 或 OUTSHR 指令时，如果就绪位被清除，则端口提取数据以及就绪位被设定。如果就绪位被设定，则线程中止直到就绪位被端口清除。

[0135] 为了在两个线程间通信，需要分配两个通道末端，每个线程一个。这是用 GETR CHAN 指令执行的。两个线程可使用资源标识符通过使用输出和输入指令来传输数据字：

[0136] OUT $d \lhd s$

[0137] IN $d \lhd s$

[0138] 如果输出指令是在通道太满而不能提取数据时执行的，则执行指令的线程被中止。当通道中有足够空间用于成功完成指令时，继续执行指令。类似地，当输入指令被执行以及有足够的可用数据时，线程暂停以及当有足够数据可用时重新开始执行。当不再需要时，可用 FREE CHAN 指令释放通道。否则通道可用于其他信息。

[0139] 事件和中断允许资源（端口和通道）自动将控制发送到预定的事件控制器 (handler)。线程接受事件或中断的能力是由保存在线程状态寄存器 SR 中的信息来控制的（参看图 4），以及可明确地用 TSE 和 TSD 指令控制。该信息包括事件启用标记 (EE) 和中断启用标记 (IE)。

[0140] TSE s SR ← SR $\vee$ s 线程状态启用

[0141] TSD s SR ← SR $\wedge \neg$ s 线程状态禁止

[0142] 这些指令的操作数应是以下之一：

[0143] EE 启用或禁止事件

[0144] IE 启用或禁止中断

[0145] 事件是在其被建立的同一范围内被处理。因此，对于所有线程的状态都有效的事件，允许线程对事件快速反应。线程可以使用引起事件同时留下某些或所有事件信息不变的端口执行输入和输出操作。这允许线程完成处理事件并立即等待另一个类似事件。

[0146] 事件控制器的程序位置必须在用 SETV 指令启用事件前设定。端口具有确定何时发生事件的条件，这些是用 SETC 和 SETD 指令设定的。通道只要含足够的数据或具有接受要输出数据的空间就被视为就绪。

[0147] 可以用事件启用无条件 (EEU) 指令来启用特定端口或通道的事件生成，以及使用事件禁止无条件 (EDU) 指令将其禁止。如果事件条件操作数为真，则事件启用真 (EET) 指令启用事件，否则禁止事件；相反如果事件条件操作数为假，则事件启用假 (EEF) 指令启用事件，否则禁止事件。这些指令被用来优化受保护输入 (guarded inputs) 的执行。下面是配置端口上事件的某些示例指令格式，但可以理解同样的指令可针对通道使用。

[0148] SETV port, v port[vector] ← v 设定事件矢量

[0149] SETD port, d port[data] ← d 设定事件数据

[0150] SETC port, c port[ctrl] ← c 设定事件控制

[0151] EET port, b port[enable] ← b ; port[tid] ← thread 事件启用真

[0152] EEF port, b port[enable] ← $\neg$ b ; port[tid] ← thread 事件启用假

[0153] EDU port port[enable] ← false ;port[tid] ← thread 事件禁止

[0154] EEU port port[enable] ← true ;port[tid] ← thread 事件启用

[0155] 已经启用一个或多个资源上的事件后，线程可使用 WAITEU 指令等待至少一个事件。这可导致事件立即发生，且控制被发送到事件控制器，该事件控制器由相应事件矢量指定，且事件是通过清除 EE（事件启用）标记禁止的。可选地，线程可暂停，直到事件发生——在该情形中，EE 标记将在事件发生时被清除，以及线程继续执行。

[0156] WAITET b if b then SR[EE] ← true 如果真则事件等待

[0157] WAITEF b if ¬ b then SR[EE] ← true 如果假则事件等等

[0158] WAITEU SR[EE] ← true 事件等待

[0159] CLRE SR[EE] ← false ; 禁止所有事件

[0160] 对所有端口 对线程

[0161] if port[tid] = thread then port[enable] ← false

[0162] 为了优化重复等待一个或多个事件直到条件发生的常见情形，提供了事件等待指令的条件形式。WAITET 指令仅在其条件操作数为真时等待，且 WAITEF 仅在其条件操作数为假时等待。

[0163] 线程启用的所有事件可用单个 CLRE 指令禁止。这会禁止具有由线程所启用的事件的所有端口中生成事件。CLRE 指令也清除线程状态寄存器中的事件启用状态。

[0164] 为了优化线程对高优先级资源的响应性，在随后开始启用端口和 / 或通道以及使用一个事件等待指令前，TSE EE 指令可用来首先启用线程上事件。这样，处理器可按照优先级顺序扫描资源。这可以使事件在被启用时立即被处理。

[0165] 与事件相比，中断不是在当前范围内处理的，因此当前 PC 和 SR（和潜在的某些或所有其他寄存器）必须在中断控制器执行前被保存。一旦资源 r 生成中断，自动发生以下动作：

[0166] SAVEPC ← PC ;

[0167] SAVESR ← SR ;

[0168] SR[EE] ← false ;

[0169] SR[IE] ← false ;

[0170] PC ← r[vector]

[0171] 在控制器完成时，中断的线程的执行可由 RFINT 指令执行。

[0172] RFINT PC ← SAVEPC ;从中断返回

[0173] SR ← SAVESR

[0174] 当线程被暂停以等待事件时，一个中断可中断线程。

[0175] 下面的例子示出了指令是如何被线程用来执行输入，输出和逻辑操作的。在例子中，使用下面的指令：

[0176] LDFI :加载指令地址到寄存器中

[0177] LDI :加载常数值到寄存器中

[0178] EQI :如果寄存器值等于常数，则产生布尔 (Boolean)（真）值

[0179] OR :产生两个寄存器值的逻辑或 (OR)

[0180] ADD :将两个寄存器值相加

[0181] ADDI :将常数加到寄存器值

[0182] SHL :向左移位寄存器内容

[0183] BBF :如果布尔值为假,则跳转到程序中另一个点

[0184] OUT :输出数据

[0185] 下面示出从引脚串行输入 8 位字节的示例代码。当在第二端口从外部时钟接收的信号从 0 变为 1(表示应提取数据) 时,数据的每个位从第一端口输入。在高级语言中,该操作看起来像这样 :

```
[0186] PROC inbyte(clock,data)
[0187] VAR byte ;
[0188] {FOR n = 0 FOR 8
[0189]    WHEN clock ? TO 1 DO{data ? bit ;byte = (byte << 1)+b} ;
[0190] RETURN byte
[0191] }
```

[0192] 下面示出该过程的指令级程序。

```
[0193] SETC    clock,TRANSITION
[0194] SETD    clock,1
[0195] LDI     byte,0
[0196] LDI     count,0
[0197] loop :
[0198] IN      clock,c
[0199] IN      data,bit
[0200] SHL     byte,byte,data
[0201] ADD     byte,byte,bit
[0202] ADDI    count,count,1
[0203] EQI     go,count,8
[0204] BBF     go,Loop
```

[0205] 通过将每个代码序列分配给其自身线程,可以同时执行两个或多个这类代码序列。

[0206] 下面示出示例代码,使用部分上述指令执行 NAND 类型的进程,该类型的进程每当两个输入 x 和 y 中的一个改变状态时醒来 (wake up)。高级代码是 :

```
[0207] PROC nand(x,y,z)IS
[0208] WHILE TRUE
[0209] {WHEN x ? TO notx DO{notx: = NOT notx ;z ! (notx ORnoty)}
[0210] |WHEN y ? TO noty DO{noty: = NOT noty ;z ! (notx ORnoty)}
[0211] }
```

[0212] 在低级代码中,该进程包括分别用矢量“xv”和“yv”初始化两个端口 x 和 y 的单个线程,并启用这些端口以生成事件。相应指令级程序如下 :

```
[0213] nand:
[0214] LDI    notx,1
```

[0215] LDI noty,1

[0216] SETC x,TRANSITION

[0217] SETD x,notx

[0218] SETC y,TRANSITION

[0219] SETD y,noty

[0220] LDFI temp,xv

[0221] SETV temp,x

[0222] LDFI temp,yv

[0223] SETV temp,y

[0224] EEU x

[0225] EEU y

[0226] WAIT

[0227] xv:

[0228] EQI notx,notx,0

[0229] SETD x,notx

[0230] OR temp,notx,noty

[0231] OUT z,temp

[0232] WAIT

[0233] yv:

[0234] EQI noty,noty,0

[0235] SETD y,noty

[0236] OR temp,notx,noty

[0237] OUT z,temp

[0238] WAIT

[0239] 操作中,x 输入改变或 y 输入改变,以及控制发送到 xv 或 yv。在任一情形中,响应代码 (response code) 执行 5 个指令,然后等待下一个输入状态变化。从输入变化到输出变化的等待时间可小于约 10 个周期。1GHz 处理器可模拟 100MHz 逻辑。

[0240] 作为另一个例子,下面示出执行 D 型触发器逻辑的进程,该进程每当输入改变状态 (但仅在由外部时钟计时时改变输出) 时醒来。高级程序是 :

[0241] PROC dtype(d,ck,q)IS

[0242] WHILE TRUE

[0243] {WHEN d ? TO notd DO notd: = NOT notd

[0244] |WHEN ck ? TO TRUE DO q ! NOT notd

[0245] }

[0246] 相应指令级程序是 :

[0247] dtype:

[0248] IN d,temp

[0249] EQI temp,temp,0

[0250] SETC d,TRANSITION

[0251] SETD d,temp

[0252] SETC ck TRANSITION

[0253] LDI temp,1

[0254] SETD ck,temp

[0255] LDFI temp,dv

[0256] SETV d,temp

[0257] LDFI temp,ckv

[0258] SETV ck,ckv

[0259] EEU d

[0260] EEU ck

[0261] WAIT

[0262] dv:

[0263] IN d,temp

[0264] SETD d,temp

[0265] WAIT

[0266] ckv:

[0267] EQI temp,notd,0

[0268] OUT q,temp

[0269] WAIT

[0270] 操作中,要么 d-input 改变,要么 ck-input 改变。在任一情形中,响应代码执行 3 条指令,然后等待下一输入状态变化。从输入变化到输出变化的等待时间可小于约 10 个周期。此外,1GHz 处理器可模拟 100MHz 逻辑。

[0271] 下面给出一些更复杂逻辑的例子。类似于 D 型,其跟踪输入数据(可以是几个位宽),以便在外部时钟到达时建立该逻辑(另一种方式仅读取时钟数据,该情形中数据可有非零保存时间)。下面的例子中通过查询表以及时钟输出计算了输出。可计算输入的更复杂函数,以及这会在下面指示的点潜在地增加更多指令。然而,注意到处理器可以只用几条指令来计算某些非常复杂的函数(相对小 LUT)。高级代码是:

[0272] PROC lookup(d,ck,q)IS

[0273] WHILE TRUE

[0274] {WHEN d ? x:x ! = lastx DO lastx: = x

[0275] |WHEN ck ? TO TRUE DO q ! lookup[lastx]

[0276] }

[0277] 相应指令级程序是:

[0278] logic:

[0279] IN d,data

[0280] SETC d,NE

[0281] SETD d,data

[0282] SETC ck,TRANSITION

[0283] LDI temp,1

[0284] SETD ck,temp

[0285] LDFI temp,dv

[0286] SETV d,temp

[0287] LDFI temp,ckv

[0288] SETV ck,temp

[0289] EEU d

[0290] EEU ck

[0291] WAIT

[0292] dv:

[0293] IN d,data

[0294] SETD d,data

[0295] WAIT

[0296] ckv:

[0297] LDW temp,lookup,data// 将其取代从而改变函数

[0298] OUT temp,q

[0299] WAIT

[0300] 在操作中,要么 d-input 改变,要么 ck-input 改变。在任一情形中,响应代码执行 3 条指令,然后等待下一输入状态变化。从输入变化到输出变化的等待时间可小于约 10 周期。此外,1GHz 处理器可模拟 100MHz 逻辑。

[0301] 还需注意的是,上面的例子展示了给定线程如何处理多个活动,诸如多个事件。

[0302] 与事件相比,中断要求状态保存在中断控制器的入口以及在出口恢复,为了使寄存器在控制器内可用。此外,控制器通常需要从其最后一次进入处开始检索状态以及将其保存以便下一次进入。中断控制器的简单例子在下面示出。其使用某些额外指令 :

[0303] LDWSP 用栈指针从存储器加载值

[0304] STWSP 用栈指针将值存储在存储器中

[0305] LDWDP 用数据指针从存储器加载值

[0306] STWDP 用数据指针将值存储在存储器中

[0307] EXTSP 用来扩展堆栈从而为新值留出空间

[0308] LDAWSP 用来从堆栈中丢弃值

[0309] 该例子输入数据字节,一次一位 ;与上面使用事件的例子相比,该例子使用中断控制器。高级程序是 :

[0310] PORT clock:INT,TRANSITION,1 ;

[0311] VAR byte ;

[0312] VAR count ;

[0313] byte: = 0 ;

[0314] count: = 0 ;

[0315] ENABLE clock ;

[0316] // 要中断的程序跟在此处之后

[0317] HANDLER inbyte()

[0318] {data ? bit ;byte: = (byte << 1)+bit ;

[0319] count: = count+1 ;

[0320] IF count = 8 THEN DISABLE clock

[0321] }

[0322] 当端口被启用以生成中断时，每次外部时钟过渡到逻辑 1 时都进入中断控制器。控制器提取数据位并且形成字节。该字节与几个位输入一起存储在存储器中的位置上以及经数据指针进行存取。当 8 位已经输入时，控制器禁止另外的中断，以便字节准备好供程序使用。相应指令级程序为 ：

[0323] SETD clock,1

[0324] SETC clock,TRANSITION

[0325] SETC clock,INT

[0326] LDI r0,0

[0327] STWDP r0,byte

[0328] STWDP r0,n

[0329] EEU clock

[0330] ...// 将被中断的程序

[0331] int:// 中断控制器

[0332] EXTSP 2

[0333] STWSP R0,0

[0334] STWSP R1,1

[0335] IN clock,r0

[0336] IN data,r0

[0337] LDWDP r1,byte

[0338] SHL byte,byte,1

[0339] ADD byte,byte,r0

[0340] STWDP r1,byte

[0341] LDWDP r0,n

[0342] ADDI r0,r0,1

[0343] STWDP r0,n

[0344] EQI r0,r0,8

[0345] EEF clock,r0

[0346] LDWSP R1,1

[0347] LDWSP R0,0

[0348] LDAWSP 2

[0349] RFINT

[0350] 从上面的说明和例子可以看出各端口的活动如何与各线程相关联，以及如何基于该活动引起的事件调度这些线程，有利地提供了可对外部激励快速响应的处理器。

[0351] 可以理解上面的实施例仅以例子说明的。在其他实施例中，可根据所需芯片规范提供不同寄存器组和指令。在某些实施例中，线程标识符不必发送到端口，而是可保留线程

调度器的责任或存储在其他地方。可选地，可以在端口为每个线程提供个别的就绪标记，以便线程标识符被发送到端口从而选择正确的就绪信号，但线程标识符不必在检测到活动后返回线程调度器。另外，条件和 / 或条件数据不必发送到端口。而是条件可在端口预配置以及 / 或者条件可在线程调度器或其他地方被评估。可以基于来自除了端口和通道的其他源的活动来调度线程。不同互连可提供在处理器不同组件之间。而且本发明不专用于具有移动应用处理器的移动终端。其他应用和配置对本领域技术人员是显然的。本发明的范围不是由所述实施例，而是由权利要求限定的。

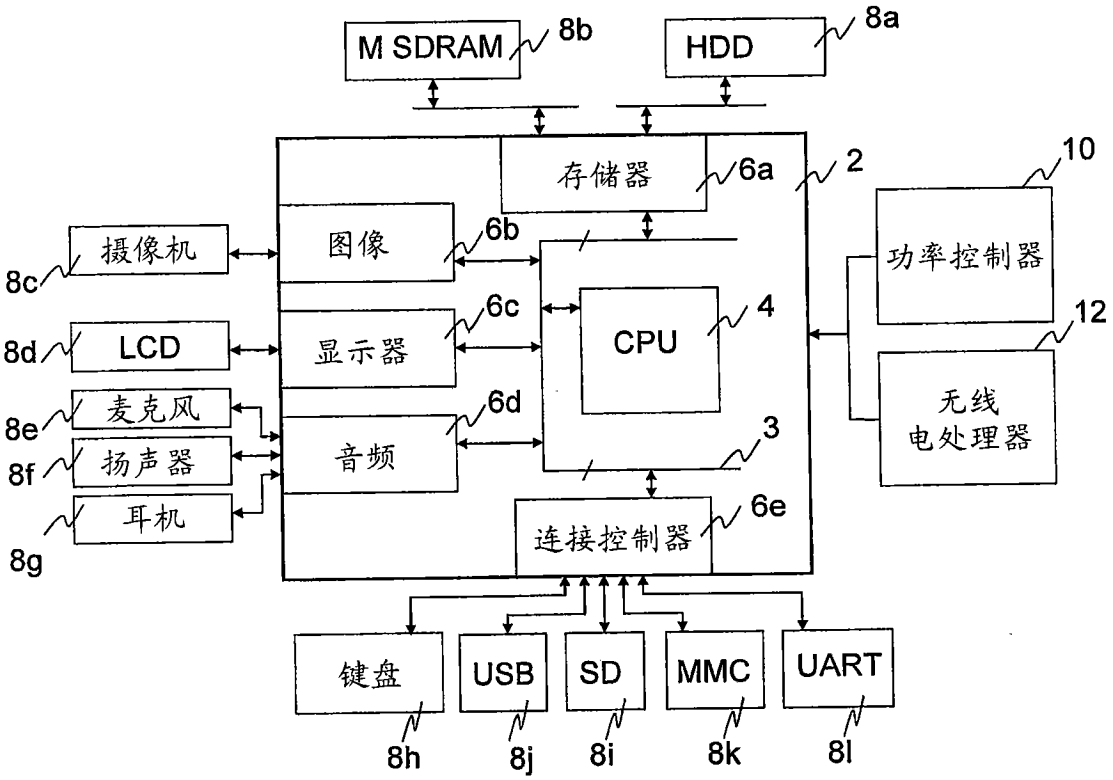

M SDRAM
8b
HDD
8a
存储器
6a
2
10
图像
6b
摄像机
8c
功率控制器
CPU
4
6c
显示器
LCD
8d
12
无线
电处理器
麦克风
8e
6d
音频
3
扬声器
8f
耳机
8g
连接控制器
6e
键盘
USB
SD
MMC
UART
8h
8j
8i
8k
8l

图 1

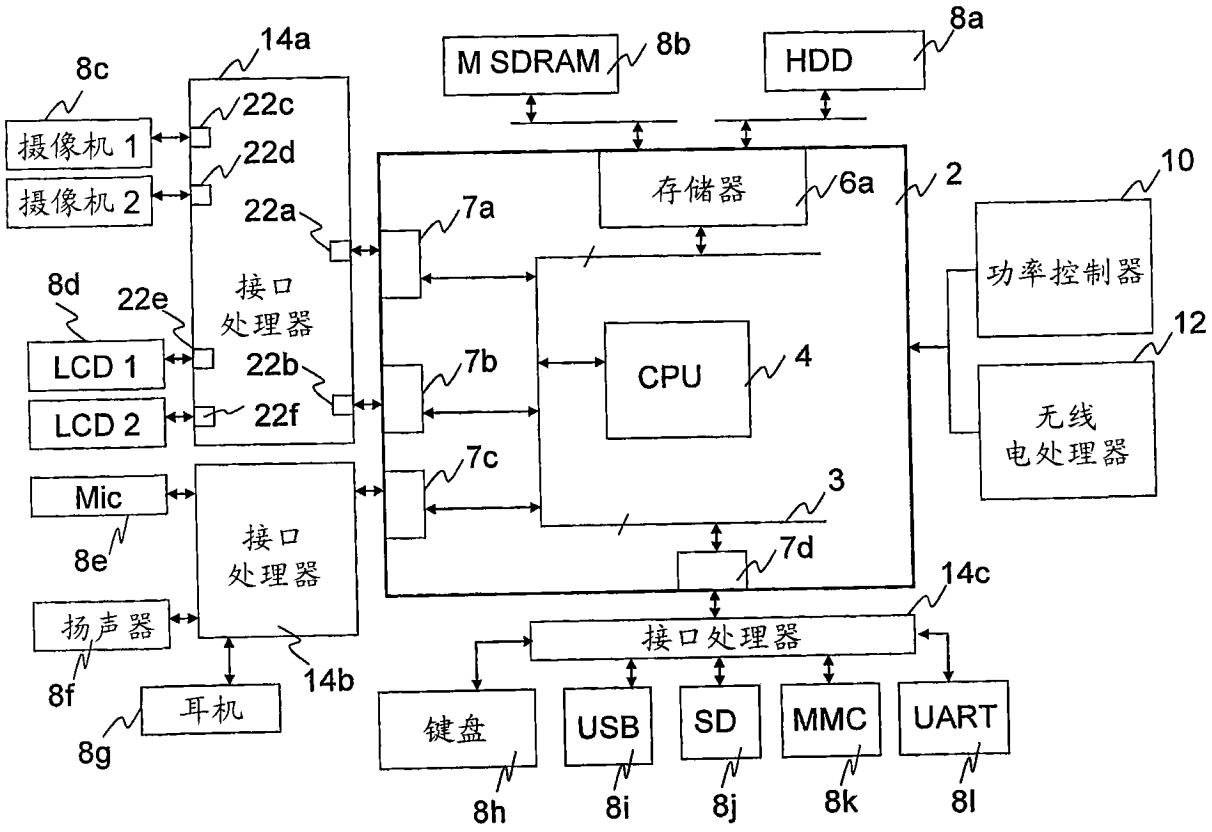

14a
8c
M SDRAM
8b
HDD
8a
摄像机 1
22c
22d
摄像机 2
存储器
6a
2
10
22a
7a
接口
处理器
功率控制器
8d
22e
12
LCD 1
22b
7b
CPU
4
LCD 2
22f
无线
电处理器
7c
Mic
3
接口
处理器
8e
7d
14c
扬声器
接口处理器
14b
8f
耳机
键盘
USB
SD
MMC
UART
8g
8h
8i
8j
8k
8l

图 2

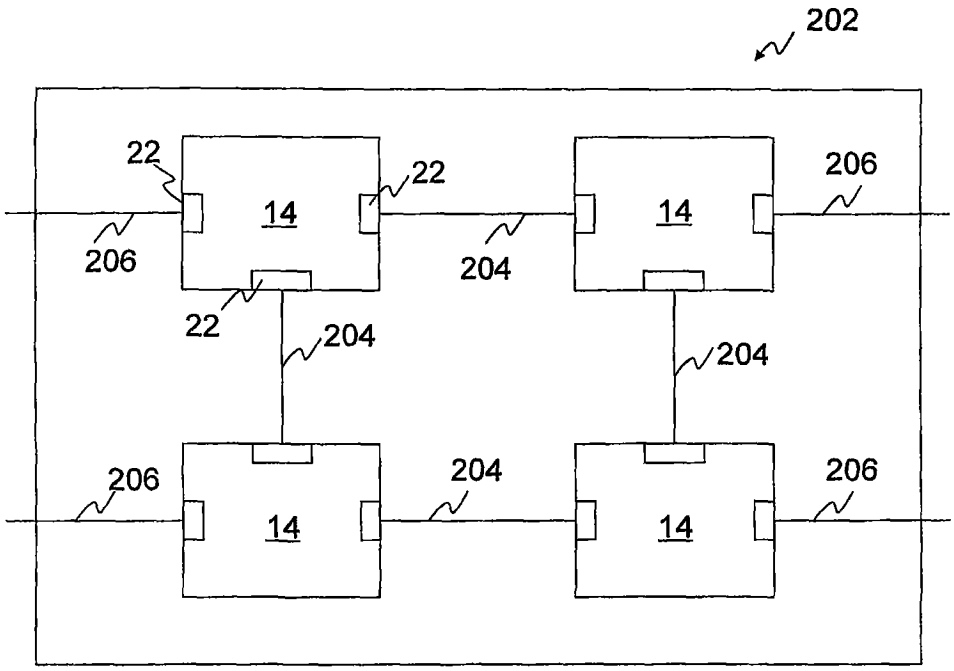

202
22
22
206
14
14
206
206
204
22
204
204
206
204
206
14
14

图 2A

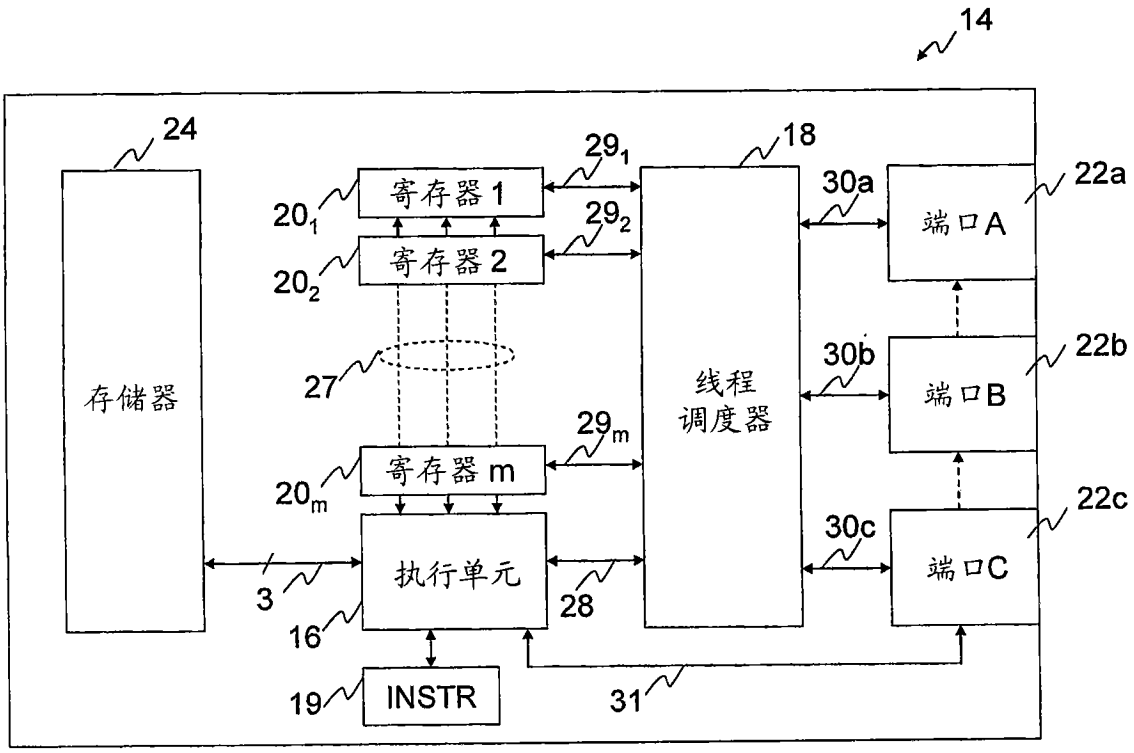

14
24
存储器
寄存器 1
寄存器 2
寄存器 m
20_1
20_2
20_m
29_1
29_2
29_m
27
18
线程
调度器
30a
30b
30c
端口A
端口B
端口C
22a
22b
22c
执行单元
3
16
28
19
INSTR
31

图 3

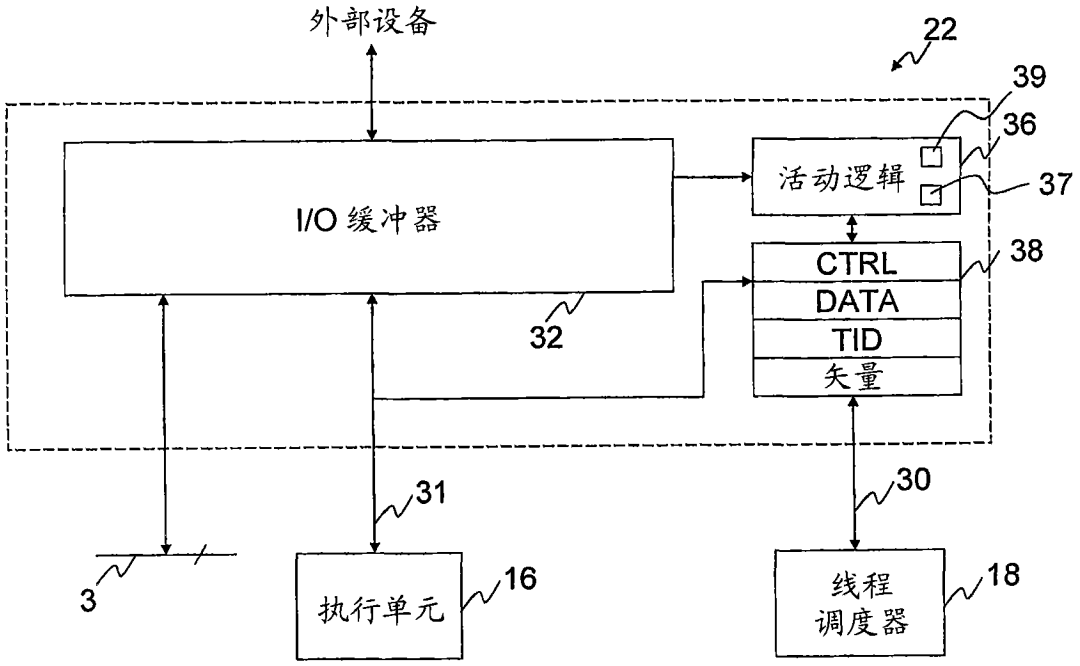

外部设备
22
39
36
37
I/O 缓冲器
活动逻辑
CTRL
DATA
TID
矢量
38
32
31
30
3
执行单元
16
线程
调度器
18

图 4

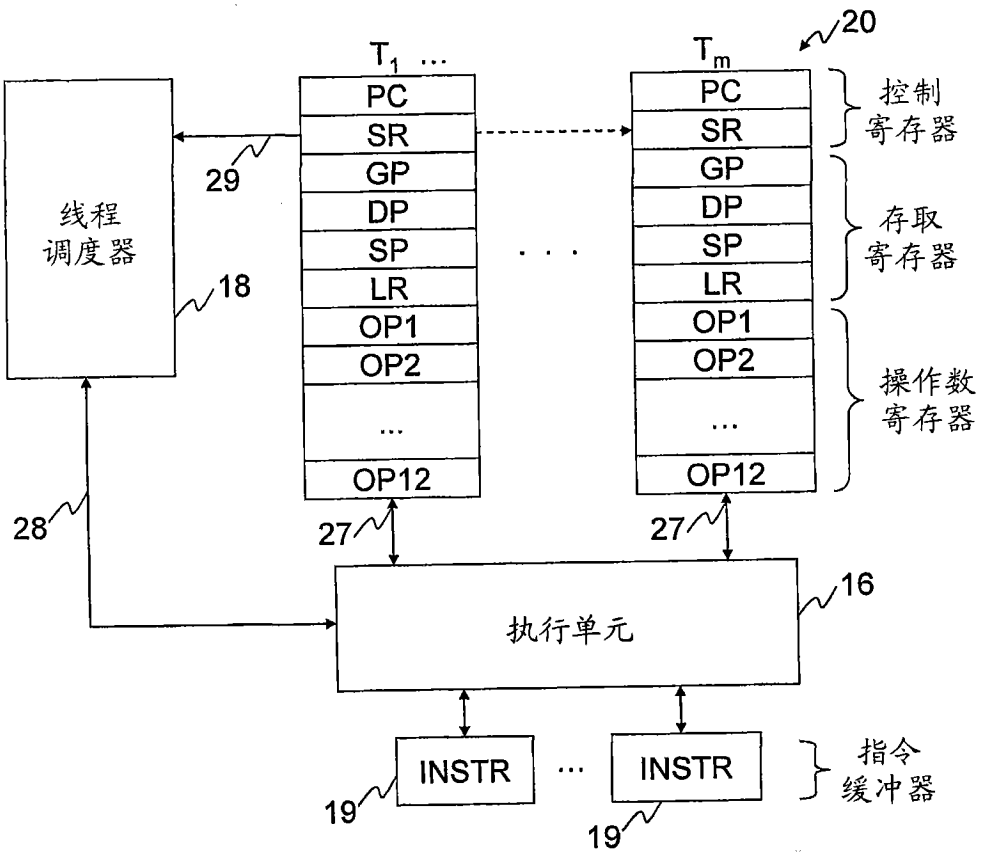

20
T_1 ...
T_m
PC
SR
GP
DP
SP
LR
OP1
OP2
...
OP12
控制寄存器
存取寄存器
操作数寄存器
线程调度器
18
29
28
27
27
执行单元
16
INSTR
...
INSTR
19
19
指令缓冲器

图 5

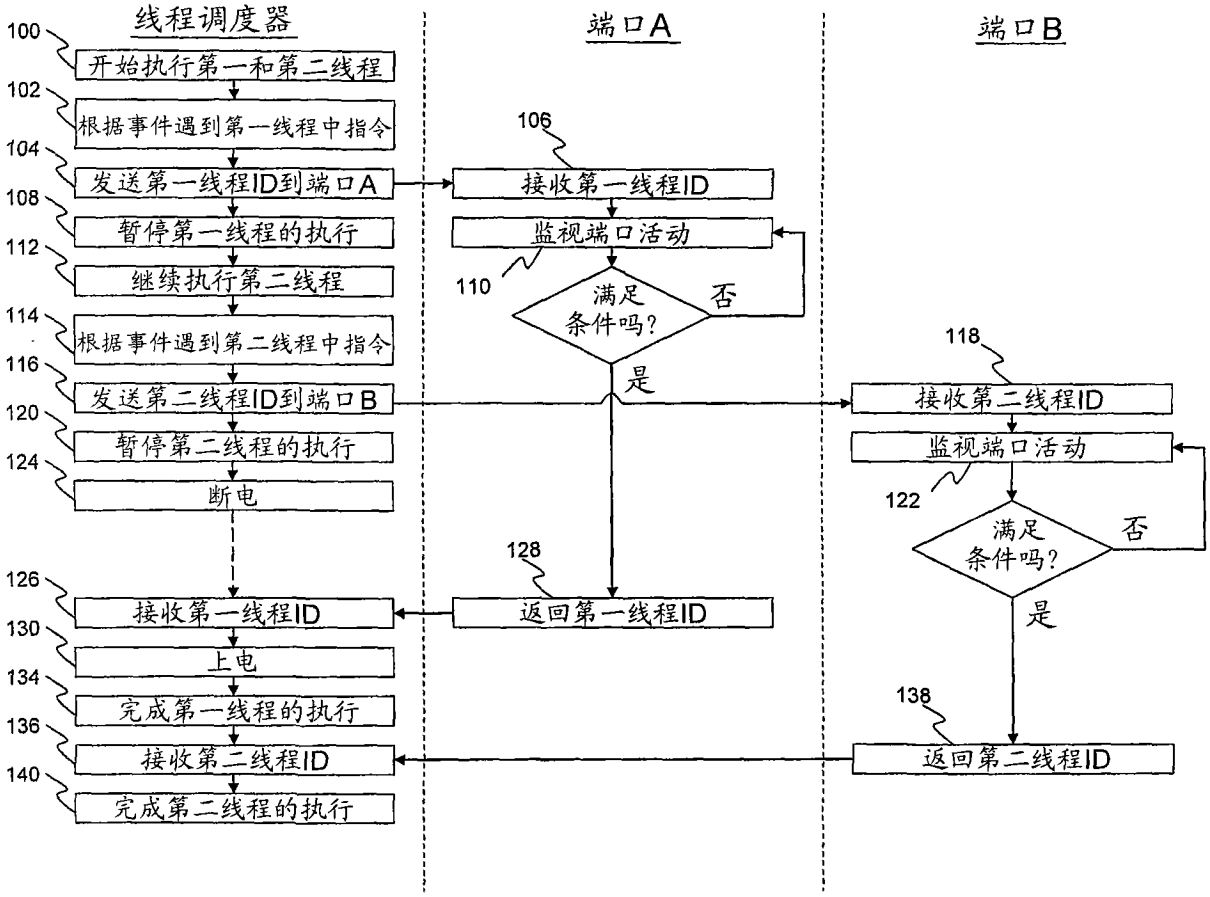

线程调度器
端口A
端口B
100
开始执行第一和第二线程
102
根据事件遇到第一线程中指令
104
发送第一线程ID到端口A
108
暂停第一线程的执行
112
继续执行第二线程
114
根据事件遇到第二线程中指令
116
发送第二线程ID到端口B
120
暂停第二线程的执行
124
断电
126
接收第一线程ID
130
上电
134
完成第一线程的执行
136
接收第二线程ID
140
完成第二线程的执行
106
接收第一线程ID
监视端口活动
110
满足条件吗?
否
是
128
返回第一线程ID
118
接收第二线程ID
监视端口活动
122
满足条件吗?
否
是
138
返回第二线程ID

图 6

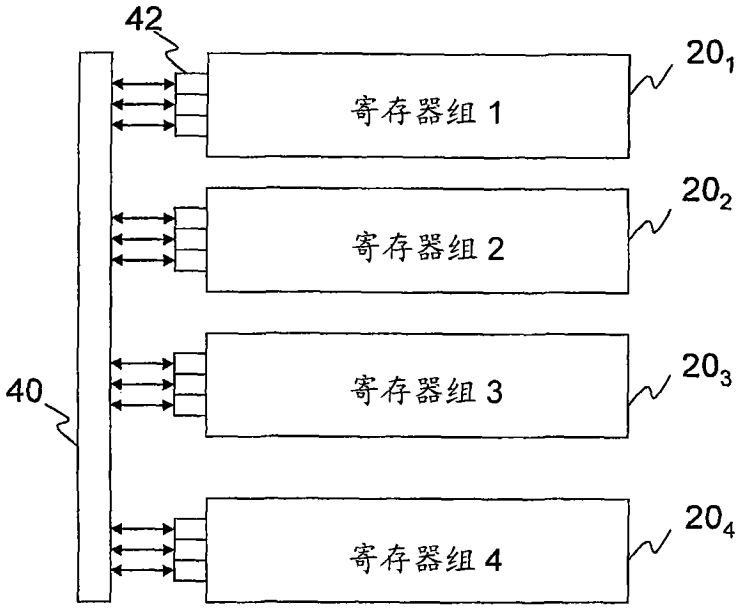

42
40
寄存器组 1
寄存器组 2
寄存器组 3
寄存器组 4
20₁
20₂
20₃
20₄

图 7

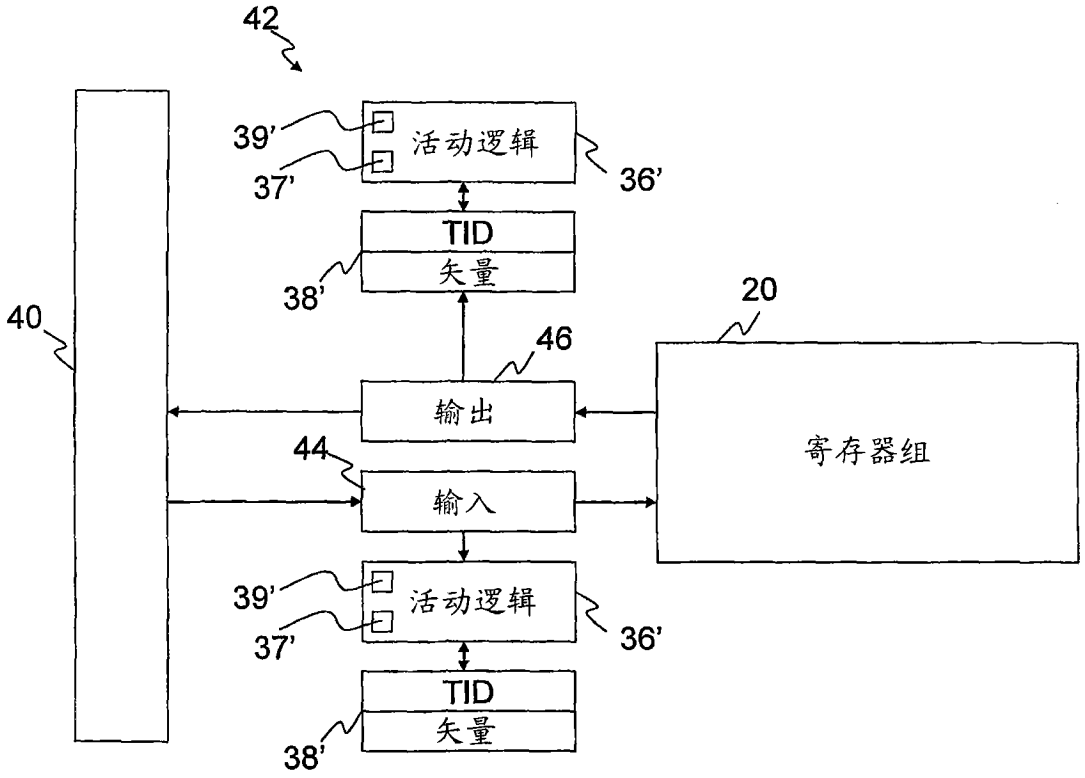

42
39'
37'
活动逻辑
36'
TID
矢量
38'
40
20
46
输出
寄存器组
44
输入
39'
37'
活动逻辑
36'
TID
矢量
38'

图 7A