PCI Mobile Design Guide

Version 1.1

December 18, 1998

Revision History

Revision	Issue Date	Comments
1.0	October 27, 1997	Original Issue.
1.1	December 18, 1998	Various edits and updates.

DISCLAIMER

This PCI Mobile Design Guide is provided "as is" with no warranties whatsoever, including any warranty of merchantability, noninfringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. The PCI SIG disclaims all liability for infringement of proprietary rights, relating to use of information in this specification. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

All product names are trademarks, registered trademarks, or servicemarks of their respective owners.

CONTENTS

7. Use of Mixed Clock Run Support Components

8. PCI Buffer Leakage Control

FIGURES

TABLES

Preface

The original Mobile Design Guide was published on October 27, 1994 and addresses the issue of managing power in the platform. At the time Mobile platforms were the primary focus of the effort. Since that time the concepts of managing the power consumed by devices that reside on PCI has evolved to include platforms from Mobile platforms to Desktop systems and is moving toward servers. As this evolution has taken place, two formal specifications have been developed to meet these needs: the *Advanced Configuration and Power Interface* (ACPI) (http://www.teleport.com/~acpi/) and the *PCI Power Management Interface Specification* (PCI-PM) (http://www.pcisig.com). ACPI is focused at the platform devices while PCI-PM is focused at the PCI add-in card market. ACPI requires the BIOS to know and control the devices while the PCI-PM specification defines a common programming interface define in PCI configuration space.

This version of the Mobile Design Guide has been updated to reflect these changes in the market. Some of the information that was included in the original version has been updated and incorporated into other specifications and, therefore, has been removed from this version. When this is done, a reference to the updated material has been added. The remaining information is of a tutorial nature about the concepts of power management for a device. The exception to this is the definition of **CLKRUN#**. It has been included in this Design Guide since the **CLKRUN#** signal is included in the PCI Local Bus Specification to help facilitate the development of common silicon for both PCI and CardBus designs. (Note that **CLKRUN#** is not supported in the PCI connector and is viewed as a sideband signal.)

Terminology

The following list contains definitions of terms used throughout this document.

agent	A logical entity that operates on a computer bus. The term applies collectively to functions of a bus master or a bus slave, or to a combination of both.
CardBus	The 32-bit PC Card (PCMCIA) interface. Loosely used, the term CardBus may refer to either the expansion slot in the system, or to the interface specification itself.
CardBus controller	The chip (or chips) that isolates the CardBus PC Cards from the rest of the system. Depending on the definition of the system bus, this might be a set of electrical buffers or a complete bus bridge. Same as host CardBus adapter or host CardBus bridge.
clock status	As a variation on the clock run protocol, the clock may be slowed to a non-operational frequency instead of being fully stopped. This is a system dependent option. PCI Mobile agents / devices must support a full clock stop.
host	An agent or a group of agents which control system setup and configuration, and system resource management, same as system master from the CardBus point of view.
host CardBus bridge	The chip (or chips) that isolates the CardBus PC Cards from the rest of the system. Depending on the definition of the system bus, this might be a set of electrical buffers or a complete bus bridge. See CardBus controller.
insertion/removal	Cold: the socket is powered down (Vcc), signals are tristated. Warm: the socket is powered up (Vcc), signals are tristated. Hot: the socket is powered up (Vcc and signals).
master	An agent that has an ability to obtain control of the interface and perform memory or I/O reads and writes to system resources.
non-operational frequency	This is a frequency of **CLK** where normal operations and data transfers do not occur across a PCI interface. This is indicated by **CLKRUN#** in a high state. The **CLK** will either be stopped or running very slowly.
operational frequency	This is a frequency of **CLK** where normal operations and data transfers occur across a PCI interface. This is indicated by **CLKRUN#** in the low state. The operational frequency may be a range of frequencies and is not guaranteed to be the maximum frequency available in the system.
PCI agent	An electrical component conforming to the *PCI Local Bus Specification* for operation in a PCI local bus environment.
PCMCIA	Personal Computer Memory Card International Association.
power management	A system level methodology which utilizes both hardware and software techniques to prolong the battery life of a system.

Reset	An agent's default state after a power-up or a "warm" reset. PC Card agents must enter Reset state upon insertion of the Card into a socket. Power-up reset: configuration of a agent programmable by the system is not required to be preserved after a Power-up reset. Warm reset (without switching Vcc off and on): agent configuration is preserved after a warm reset.
system master	An agent or a group of agents that control system setup and configuration, and system resource management, same as host from the CardBus point of view.
target	An agent that sends or receives data under control of a bus master. There are two types of targets: I/O target -- selected by its address in the I/O address space Memory target -- selected by its address in the memory address space

1. Introduction

1.1. Purpose

PCI defines a low latency, high performance interconnect bus with a low pin count and "glue-less" interface ideal for mobile systems. This Design Guide is a handbook for designers of mobile systems and components incorporating the PCI bus. The Design Guide aids the PCI designer in implementing a PCI based component or system.

Figure 1 shows the functional elements of a typical mobile PCI-based system. This architecture demonstrates the use of PCI mobile agents on the system (local) bus, the use of a PCI compliant slave agent on the motherboard, and the use of CardBus Agents via a CardBus controller / bridge. Also, "shared silicon" agents for use in both PCI mobile and CardBus environments are shown.

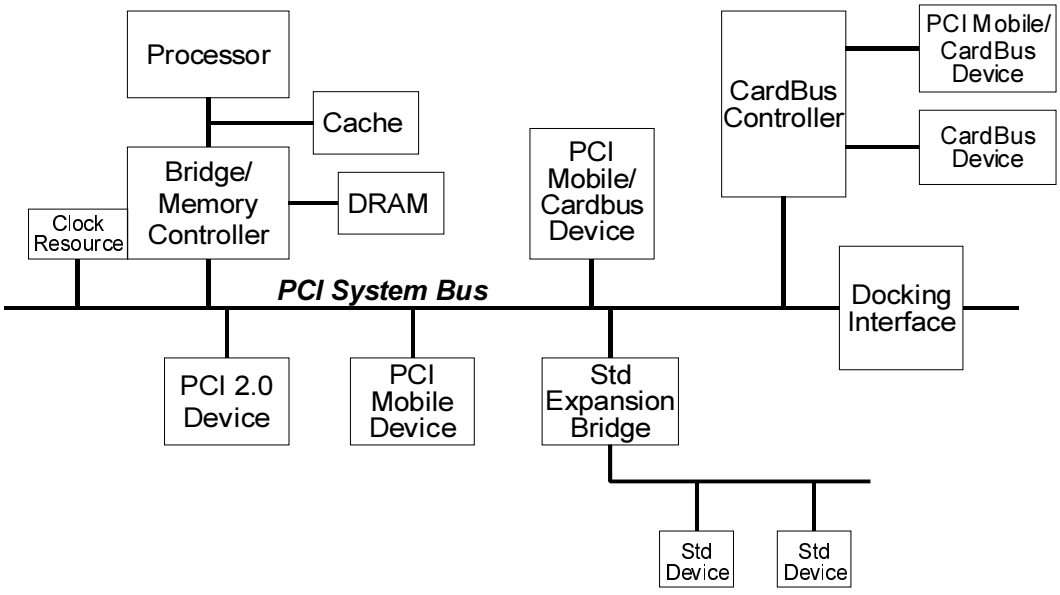

Figure 1: Functional Elements of a Typical Mobile PCI System

Computer systems of today and tomorrow are geared towards ease of user interaction and power savings in the system. To address user needs, system manufacturers have various methods in which to make the computer more appealing. Power control methods are very specific to the different system manufacturers and is most likely a proprietary solution. However, there are some general power management techniques that can be discussed. These include:

- Clock Control
- Power Partitioning and Control
- Device Control

1.1.1. Clock Control

Fundamental to computers is the ability to maintain and change the state of semiconductor devices. Most logic devices today use clocked or synchronous designs. Since any change of state of a semiconductor device consumes energy, power management techniques are sometimes used to minimize the number of unnecessary state transitions. One way to do this is to control the input clock to a logic device.

Clock control can involve either slowing the clock or stopping the clock to a given device, assuming that the device continues to function in this case. The **CLKRUN#** examples in Sections 5-8 detail a method for stopping and starting clocks in a PCI environment. These techniques may, in some cases, be extended to non-PCI system elements.

1.1.2. System Power Partitioning and Control

Since power is consumed by all active electronic devices, it should be apparent that removing power to these devices is an effective method of power management. However, it should also be apparent that this technique introduces a number of problems; that is, how is the power controlled, how does one identify when that device is needed or not needed, and how does one save and restore the state of that device if power is removed. In general, sections of a mobile product may be partitioned into one or more power partitions, in which certain devices are provided power independent of other sections.

These sections may have power applied or removed independently under certain circumstances. The process of removing and restoring power to a particular section of a mobile computer system, or to a particular device, is generally referred to as the SUSPEND/RESUME process with intermediate IDLE and STANDBY states. It implies that power is partitioned across certain boundaries, although these boundaries may be at the system, subsystem, or device levels.

It is beyond the scope of this document to detail all possible implications of power switching of mixed components. However, some of the problems that arise in a power partitioned environment are discussed. These include leakage control and SUSPEND/RESUME of PCI agents. Leakage control deals with the interaction between different logic devices when they reside in separate power partitions. SUSPEND/RESUME deals with some of the device state save and restore issues.

1.1.3. Device Control

Device control is a device management technique that limits the power consumption of a particular device or sub-system, typically a peripheral device. An example of this is the control of a diskette drive motor. A motor consumes substantial power; therefore simply turning the motor off on a diskette drive saves significant power. In DOS machines, the diskette motor is managed by BIOS; therefore the control in this case is part of the base system. The same general technique, however, can be applied to a number of devices, including hard disk motors, charge pumps, and similar devices. Since the number of peripheral devices and the specific control methods are clearly beyond the scope of this document, they are not further discussed.

Device control techniques must rely on a priori information specific to a given device or class of devices. The section on power control considerations discusses the type of information required to properly manage a device.

2. Clock Run

2.1. Clock Run Clarification / Motivation

In the *PCI Local Bus Specification*, it is stated that all components (with a few exceptions) must work with frequencies up to 33 MHz. The clock frequency may be changed at any time during the operation of the system so long as the clock edges remain "clean" (monotonic), the minimum cycle and high and low times are not violated, the PCI bus is idle, and if no bus request or lock is asserted. When the system stops the clock it must be stopped in the low state and the clock line must remain low until the clock is restarted. It does not specify a method for determining when to stop or start the clock. Since the minimum clock frequency can be as low as 0 MHz it is difficult for a PCI agent to determine if the clock is stopped or just running extremely slow.

The provisions for variable clock frequency and for stopping the clock imply that the devices interface logic must have a static design to maintain its state, and the device cannot rely on any particular frequency of the clock. Devices which are capable of bus mastership cannot assert a bus request while the clock is stopped since **REQ#** is a synchronous signal to the clock.

However, the following is not specified in the *PCI Local Bus Specification*:

- Method for determining when to stop clock.
- Method for determining when to start clock.

The **CLKRUN#** signal and its protocol answer these issues for the mobile environment. For other platforms refer to the *PCI Power Management Interface Specification* for addition information.

2.1.1. Clocking Architecture and Roles

Figure 2 and Table 1 show the relationship between agent roles in the clocking architecture.

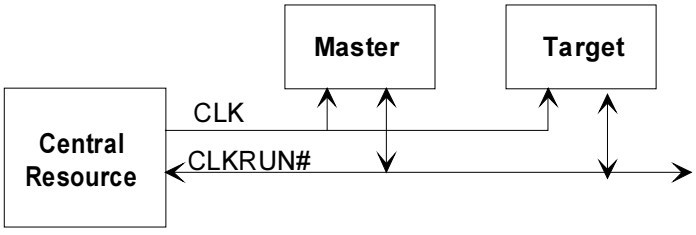

Figure 2: Clocking Agent Roles

Table 1: PCI Agent Role Definitions

Clocking Agent	Role Definition
Central Resource	This is the "clock controller" which controls modulation of the PCI clock and monitors and drives the **CLKRUN#** line to implement the clocking protocol.
Master	This is any PCI mastering agent which comprehends the clock control protocol and uses the **CLKRUN#** line to request a clock running state in order to assert **REQ#** and receive ownership of the bus.
Target	This is any PCI agent which may require PCI clocks beyond the four clocks guaranteed at the end of a transfer cycle. It uses the **CLKRUN#** line to request that the clock remain running for a period of time.

2.1.2. Signal Definition

CLKRUN# **o/d, s/t/s** *Clock Run* is an optional signal which is used by devices to request starting (or speeding up) the clock, **CLK**. **CLKRUN#** also indicates the clock status. For devices, **CLKRUN#** is an open drain output and also an input. For the central resource (provider of the clock) it is a sustained tri-state I/O signal. A device requests the central resource to start, speed up, or maintain the interface clock by the assertion of **CLKRUN#**. The central resource is responsible for maintaining **CLKRUN#** asserted, and for driving it high to the deasserted state. **CLKRUN#** is low upon deassertion of reset (since **CLK** is running upon deassertion of reset).

CLKRUN# is an input for all clocking roles, allowing the monitoring of clocking state transitions.

Output drive characteristics are defined by clocking role as follows:

Master / Target: "open drain" (that is, low drive capability only), asynchronous.

Central resource: high and low drive capability, clock synchronous when driven to a high state, along with weak controllable keeper to maintain a high level during clock stop.

2.1.3. Clocking States

There are three main states in the clocking protocol:

- **Clock Running** - the clock is running and the bus is operational
- **About to Stop/Slow Down** - the central resource has indicated on the **CLKRUN#** line that the clock is about to stop/slow down
- **Clock Stopped/Slowed** - the clock is stopped/slowed, with **CLKRUN#** being monitored for a restart

2.1.4. Operation

Figure 3 shows all of the PCI bus states and the level of the **CLKRUN#** signal for each of the corresponding Target/Master/Resource **CLKRUN#** driven state.

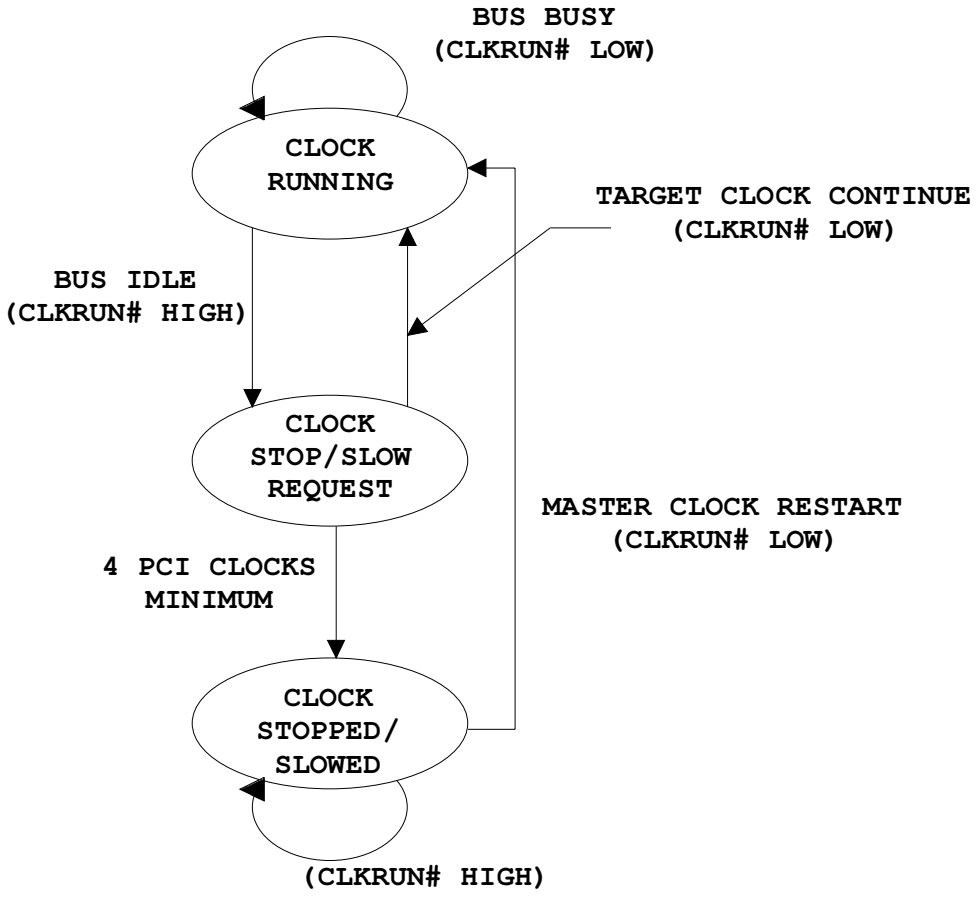

Figure 3: CLKRUN# and Bus States

2.1.4.1. Clock Stop or Slow Down

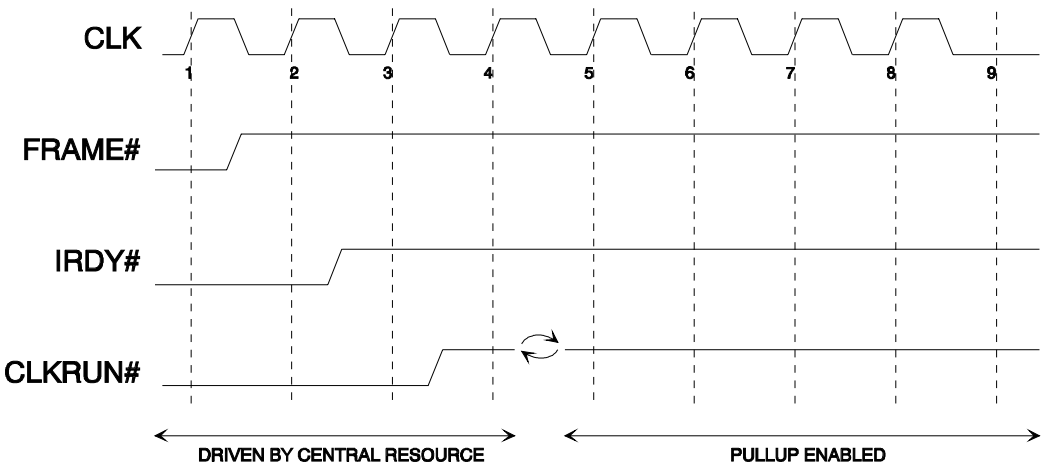

Figure 4: Clock Stop or Slow Down

The central resource drives **CLKRUN#** low while **CLK** is running at a normal operating frequency. Before stopping the clock or slowing the clock down to a non-operational frequency, the central resource synchronously drives **CLKRUN#** high for one clock period, and then tri-states its driver. A low current pull-up (a keeper) must be provided by the central resource to prevent the line from floating. Implementations may disable the pull-up when the central resource samples **CLKRUN#** low.

CLK continues to run unchanged for a minimum of four clock periods after **CLKRUN#** is deasserted. In addition, **CLK** must not be stopped before the agent can request it to continue by asserting **CLKRUN#**. For example, after clock 8, the central resource may stop or slow down the clock if all of the conditions specified above are met.

CLK is guaranteed to run unchanged for a minimum of four clock periods after **CLKRUN#** is deasserted. The central resource may keep **CLK** running unchanged for any number of clocks (greater than or equal to four).

PCI devices are required to maintain their states while the interface clock is stopped or the clock frequency is changed.

2.1.4.2. Clock Start or Speed up

Figure 5 shows that a device asserts the **CLKRUN#** signal asynchronously (since **CLK** is stopped) to request the central resource to restore **CLK**. The device holds **CLKRUN#** asserted until it detects two rising edges of **CLK**. After the second clock edge, the device must disable its open drain driver.

After detecting the assertion of **CLKRUN#**, the central resource starts the clock if the clock was stopped, or brings it to an operational frequency if the clock was slowed down.

The central resource drives **CLKRUN#** low at any time after it detects that the line is asserted by the device, but not later than on clock 3. The central resource may disable the pull-up on the **CLKRUN#** line at this time.

The central resource must not drive **CLKRUN#** high earlier than on clock 5.

The device may not assert (start driving) **CLKRUN#** if it is already driven low by the central resource. The device must not assert **CLKRUN#** unless the line has been deasserted for two successive clocks, that is, before the clock was stopped.

It is expected that a device which has asserted **CLKRUN#** for gaining bus mastership, would assert **REQ#** no later than four clocks after the clock is restarted. Otherwise, the clock can be stopped again by the central resource.

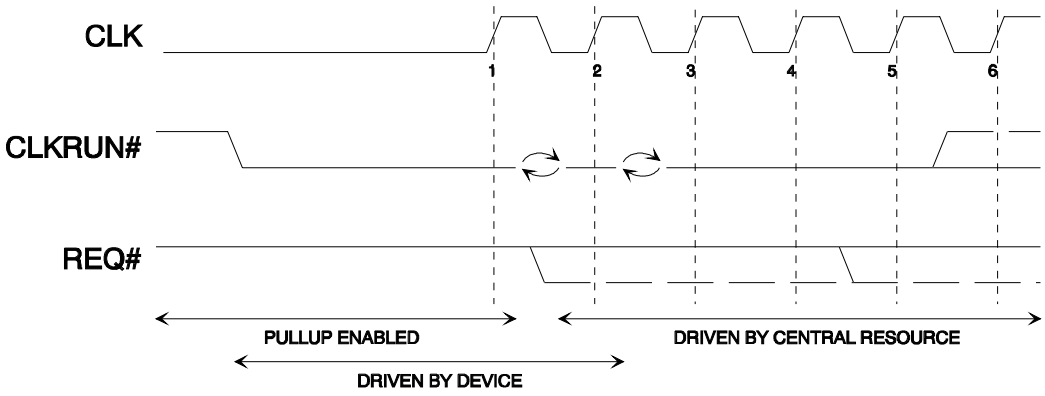

Figure 5: Clock Start or Speed up

The central resource should provide low latency clock restoration upon assertion of **CLKRUN#** (typically, not more than a few cycles of its internal clock) since this latency would negatively impact the system performance. The intent of this protocol is to provide a low latency clock control which is transparent to the system, and which would have no apparent impact on the system performance.

2.1.4.3. Maintaining the Clock

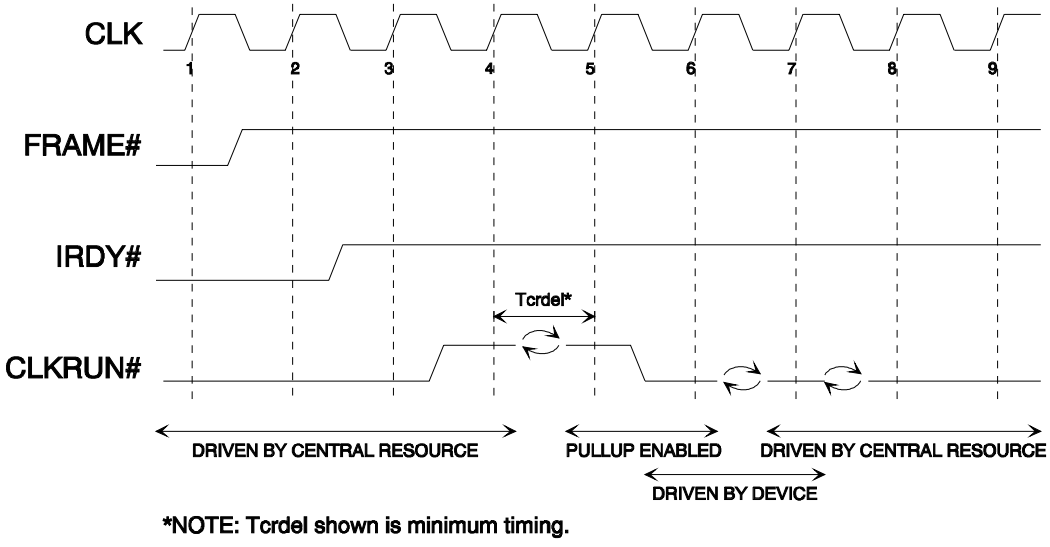

Figure 6: Maintaining the Clock

Table 2: Mininum and Maximum Values for Tcrdel

	Minimum	Maximum
Tcrdel	1 * Tcyc (1 clock)	2 * Tcyc (2 clocks)

- Certain devices may require the PCI clock to be active for completing some internal processes after a transaction is already completed. This is accomplished by the device asserting **CLKRUN#** after it has been deasserted for two successive clocks. The device must assert **CLKRUN#** within a certain time window (Tcrdel) (see Table 2 to avoid interrupting the clock stream. In Figure 6 the device samples **CLKRUN#** high on clock 4, and must drive **CLKRUN#** low no later than clock 6 but not earlier than after the turn around cycle which occurs after clock 4 to avoid interruption of **CLK**. The device keeps **CLKRUN#** asserted for two clocks (clocks 6 and 7 or clocks 7 and 8), and must disable its open drain driver after the second clock.
- The central resource must provide a non-interrupted clock when the device asserts **CLKRUN#** in the time specified above. The system designer should take into account:
  - All delays in the path to the clock controller and the clock source.
  - The time required to synchronize **CLKRUN#**. The central resource must not stop the clock before a synchronized version of the **CLKRUN#** signal from the device can be generated.
  - The central resource must drive **CLKRUN#** low no later than clock 8. The central resource may drive the line low at any time after it detects that **CLKRUN#** is asserted by the device.

The central resource must not drive **CLKRUN#** high earlier than on the fourth clock edge after the **CLKRUN#** line was first sampled asserted.

The device may not drive **CLKRUN#** if it is already driven low by the central resource. The device must not assert **CLKRUN#** unless it has sampled the line high on a **CLK** rising edge, and must not drive **CLKRUN#** on the same clock edge on which the line is first sampled high.

2.1.4.4. Clock Continuation - Minimum Repetition

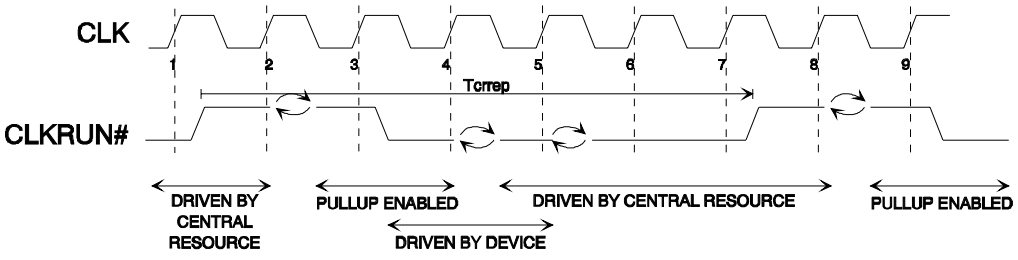

Figure 7: Multiple Clock Continues

Table 3 shows the minimum repetition rate for multiple clock continues. The case occurs when a target does a clock continue, and the central resource immediately attempts to stop the clock again. Figure 7, and the corresponding Tcrrep, specify the minimum period of repetition by the central resource and target handshake.

Table 3: Minimum Tcrrep Value

	Minimum
Tcrrep	6 * Tcyc (6 clocks)

2.1.5. Implementation Notes

2.1.5.1. Central Resource Rules

The central resource drives **CLKRUN#** low while the PCI clock is running. The central resource drives **CLKRUN#** high (synchronous to the PCI clock) before stopping the PCI clock. The PCI clock will continue to run for at least four clocks after **CLKRUN#** is driven high. **CLKRUN#** cannot be deasserted unless the bus is idle. The central resource monitors **CLKRUN#** while **CLKRUN#** is inactive, watching for asynchronous assertion of the signal. Upon seeing **CLKRUN#** asserted, the PCI clock will restart. The central resource will start driving **CLKRUN#** active to take over drive of the line. The central resource also controls the high keeper on the pin. It is suggested that the keeper is disabled by the central resource when **CLKRUN#** is being driven low by the central resource (all but one clock of the **CLKRUN#** low time). This removes active power dissipated by the keeper. The only current passing through the keeper when the clock is stopped is the device leakage current.

2.1.5.2. Master CLKRUN# Rules

A PCI master drives **CLKRUN#** low in order to restart the clock so that it can assert its bus request synchronously. Multiple masters requesting clock restart is allowed, but only one will receive a grant. After master releases drive of **CLKRUN#** the central resource takes over driving of the **CLKRUN#** signal.

A master may not assert **CLKRUN#** unless it is sampled high with **CLK** (before the clock is stopped).

2.1.5.3. Target CLKRUN# Rules

If a target of an access sample **CLKRUN#** high, it drives **CLKRUN#** low in order to maintain the clock so that it can assure internal PCI clock related functions complete. After the target samples **CLKRUN#** low for two cycles it releases drive of **CLKRUN#**. The central resource takes over low drive. In this way, the target only drives **CLKRUN#** for two **CLK** rising edges.

The target may not assert **CLKRUN#** unless it is sampled high with **CLK** (before the clock is stopped).

2.1.5.4. Clock Latency Issues

When restarting the clock after a request for clock run becomes active, it is desirable to return the clock to operational frequency as soon as possible.

An exact time duration cannot be guaranteed due to differences in system design. For example, one system may simply gate the clock between active cycles, while another may shut down the clock and its oscillator during long periods of inactivity. In the former case, the clock return can be instantaneous, while the latter case may require several milliseconds in order to stabilize the oscillator.

When determining buffer sizes for silicon targeted at portables, a portion of this clock latency should be taken into account. However, it is unreasonable to expect buffer sizes to account for re-powering of many peripherals. System designers that need to power up large portions of circuitry due to a sleep condition will have to devise another method to prevent data loss.

3. Minimum PCI Clock Frequency

In the PCI specification it is stated that the PCI clock can run at any frequency up to 33 MHz. It is also stated that the clock must be stopped in the low state. A problem arises when the clock is run very slow. Since clock Cycle time (Tcyc) maximum is infinite, the clock may be in a low or a high state for up to an infinite amount of time. For example, if the clock is run at 1/60 Hz, **CLK** will be low for 1/2 minute and high for 1/2 minute. Although this example is extreme, it is allowed under the current wording of the specification.

The minimum PCI bus clock frequency which is recommended in a Mobile PCI environment is 32 kHz. In this way, the PCI clock will only run from 33 MHz to 32 kHz and stop at DC. This changes maximum Tcyc to 32 µs if running, and infinite if stopped. The clock frequency should meet the timing requirements as specified in Table 4.

Table 4: PCI Mobile Clock Timing Requirements

Symbol	Parameter	Minimum	Maximum	Notes
Tcyc	**CLK** cycle time	30 ns	32 µs	1
Tlow	**CLK** low time	12 ns	infinite	2

Notes:
1. Maximum cycle time is enforced while **CLK** is running. If **CLK** is not running (that is, **CLKRUN#*** is not asserted), Tcyc maximum may be infinite.
2. When **CLK** is stopped, Tlow will be infinite.

Figure 8 illustrates the PCI clock timing requirements.

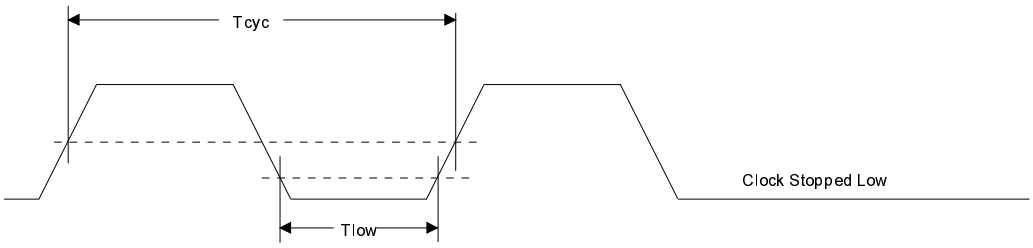

Figure 8: PCI Clock Timing Diagram

Refer to the PCI Local Bus Specification for threshold maximums and minimums.

4. PCI - CardBus/PCI Common Silicon Requirements

Refer to the CardBus/PCI Common Silicon Requirements section of the PCMCIA PC Card Standard Guidelines document to understand how one implementation can address both the CardBus and PCI Markets.

5. PCI Agent Power Capabilities

A PCI agent targeted for the mobile market must have specific power capabilities to allow a comprehensive power savings scheme. Figure 9 defines the different segments of an agent, showing three different possible segments for PCI Interface power control. The agent specific power portions are shown for completeness only and are not discussed here.

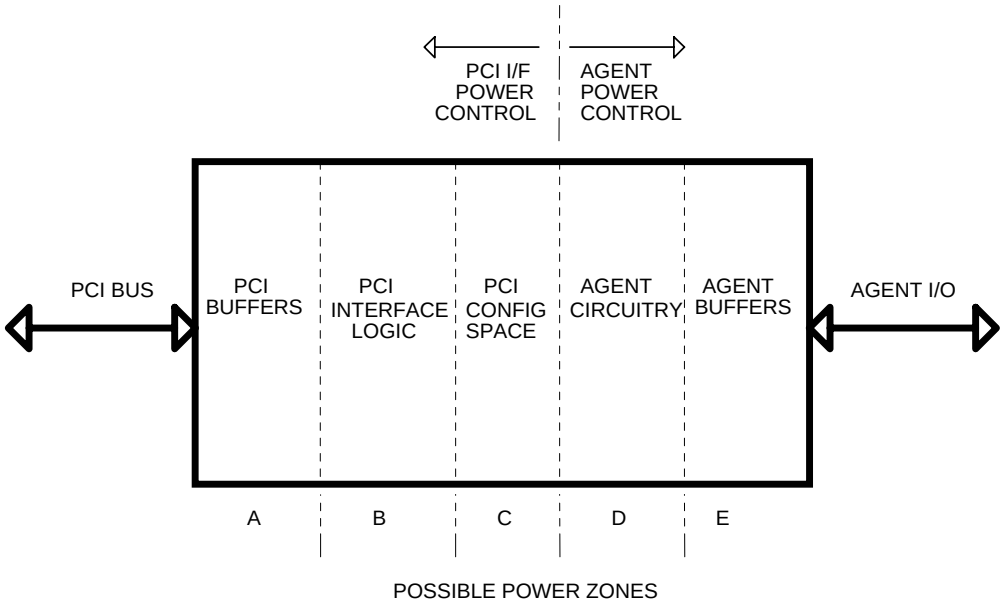

Figure 9: PCI Agent Power Segments

As shown above, an agent can be segmented into different power capability zones. Each power zone can be controlled separately to achieve maximum power savings.

The PCI interface can be thought to have many power capability levels. The buffers can be powered down while the rest of the chip is powered, saving agent parameters. The

PCI clock can be slowed or stopped, saving power used across the PCI bus and by the interface logic.

These levels range from highest power consumption to lowest. As less power is consumed, more agent parameters may be lost and latency to recover may be higher. A simple rule of thumb is shown in Figure 10, latency and power consumption directly correspond to state loss and power switching.

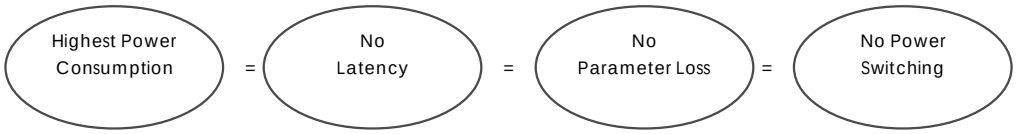

Figure 10: Latency Diagram

5.1. Clock Control

According to the *PCI Local Bus Specification*, the PCI clock operates up to 33 MHz. A variance is included for motherboard only components, which states that the component can enforce a policy of no frequency changes. For additional power savings, a designer of a mobile system can take advantage of the allowed frequency range by slowing the clock during periods of low activity and perhaps even stopping the clock during periods of inactivity (refer to Chapter 2 for discussions of agent clocking control and Section 3 for recommended frequencies). In this case, any component which enforces the policy of no frequency change should not be used.

To ensure maximum power savings, an agent must handle this variable clock frequency to be useful in a PCI Mobile environment. There are several things the agent must be capable of to do this. First, the agent should not depend on a specific frequency across the interface for proper operation. If the agent needs a specific frequency for proper operation, it should rely on a separate source for it and only use the PCI clock for interface clocking. Second, the agent should use static design practices to ensure that PCI state and configuration data is not lost when the clock is stopped.

Clocking control at a system level may be, but most frequently is not, under software control. The most optimum clocking control will be transparent to software, that is, there is no agent parameter loss and no latency to return to a normal operational frequency.

5.2. PCI Configuration and Interface Logic Capabilities

To achieve even greater power savings, the system designer removes power from idle agents while normal system activity continues. To accomplish this in a bused environment, some PCI agents remain completely active and powered, while others leave only the buffers powered. As an example, many systems disable the hard disk drive after periods of inactivity while the system continues to operate.

In this way, the PCI bus has active traffic to most frequently used agents. A form of I/O Trapping can be used to "catch" any bus transactions to the disabled agents, so that these transactions can be reissued after the agent has been restored to full operational capability.

In this scenario, configuration parameters can be lost. When the system needs this resource it will need to restore it to full operational capability, typically by reconfiguring it. The system may completely reset the agent or it may restore it to its original configuration before power off. For reconfiguration, the agent needs to make all register data available. This should be done by providing read/write-able registers, and/or shadowing write only registers. In this way, the system can read a device's pre-disabled data and restore it upon returning the device to normal activity.

5.3. Buffer Capabilities

At the highest PCI Interface power savings, the bus can be completely powered off. This can be done in conjunction with the configuration and interface logic if desired. Leakage control must be carefully planned to preserve power savings when one or more agents are powered (refer to Section 8 for discussions of agent leakage control on the PCI bus). The interface can return to full function with some latency, varying depending on power conditions. The PCI Interface may be entirely powered off while the rest of the agent is powered on. In this scenario, there is no associated state loss.

It is recognized that achieving power savings in this state is agent specific. For example, power savings means could be tightly coupled with the agent functions and the agent design, and could be determined by the agent itself or the agent's driver.

6. CLKRUN# Across a Bridge Example

When using the clock run protocol in an architecture with multiple buses and bridges, the routing of the clock and the relationship of the frequencies between the multiple buses must be considered. The *PCI to PCI Bridge Architecture Specification* states: "The relationship between the primary interface and secondary interface clocks of a PCI to PCI bridge is implementation specific." Here we show several possible implementations, with **CLKRUN#** design examples.

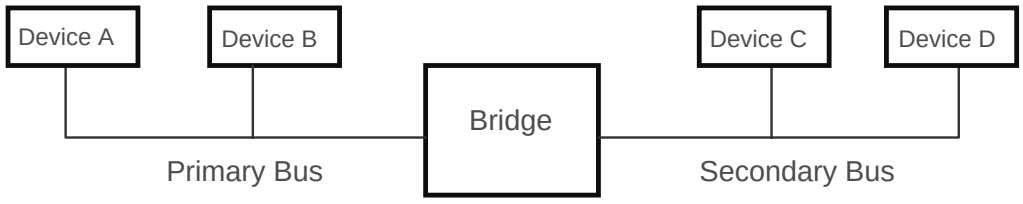

Figure 11: Bridge Architecture

Referring to Figure 11, the relationship of the operational frequency of the primary bus and secondary bus must be considered. Here, we will discuss the following relationships[1]:

- Case A: The primary bus speed is equal to the secondary bus speed. The bridge has clock buffers, introducing a slight skew, but the frequencies remain similar.
- Case B: The primary bus speed is some multiple of the secondary bus speed. The bridge has clock dividers. CardBus bridges will most likely implement this method.

[1] Refer to Section 6.4. Secondary Bus Clock Not Controlled By Bridge for a description of **CLKRUN#** handling when clocking for the secondary bus is not handled by the bridge.

- Case C: The primary bus speed has no relationship to the secondary bus speed. The primary and secondary buses are asynchronous to each other. The bridge has an external clock source for the secondary bus. This may occur when the primary bus is the host bus, running at the processor speed.

The assumption is made for simplification that the primary bus is the higher speed bus, and that the secondary bus will have its clock slowed or stopped before (or at the same time) as the primary bus.

The assumption is also made that the bridge contains the clocking resources for the secondary bus, but not the primary bus. The bridge does not drive **CLKRUN#** high on the primary bus, that is, it does not act as central resource for the primary bus. The bridge does drive **CLKRUN#** high on the secondary bus to indicate request to stop/slow the clock. The **CLKRUN#** driver (on the bridge) for the primary bus is an open-drain buffer, while the driver (on the bridge) for the secondary bus is sustained tri-state.

6.1. Case A: Skewed Clocks

When both the primary and secondary buses have the same operational frequency, **CLKRUN#** should be buffered similar to the implementation of **CLK**. The signal **CLKRUN#** is an asynchronous signal and can be buffered once without significant performance delay.

When **CLK** is simply buffered through the bridge, **CLKRUN#** can be routed around the bridge; however this is not recommended due to loading and drive requirements. The system designer must carefully plan his design, calculating buffering and timing needs. The bridge designer must give some thought to how **CLK** is to be distributed to the secondary bus so that clock control (by **CLKRUN#**) can be supported in the system.

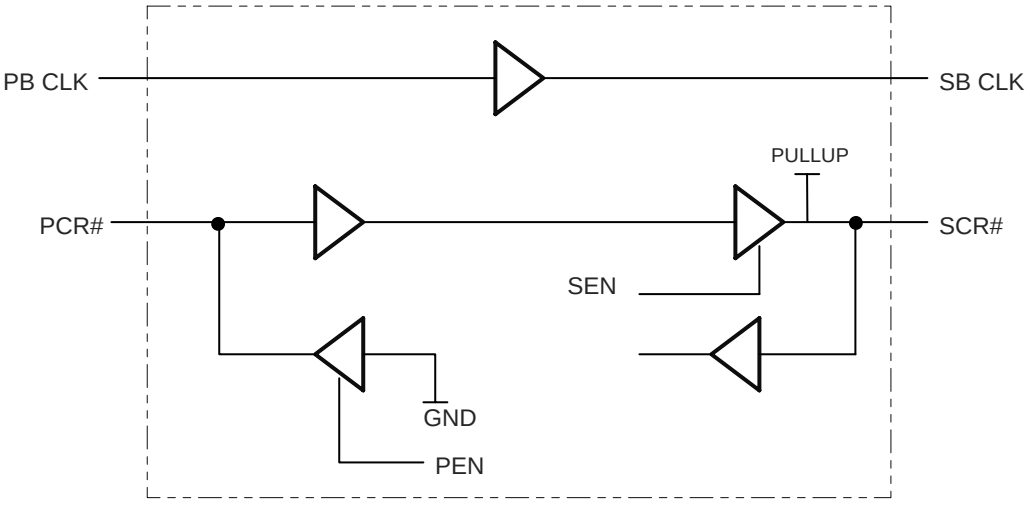

Figure 12: Bridge CLKRUN# Routing

This example highlights the details of the **CLKRUN#** protocol that need to be met by the bridge design. This is not the only possible implementation. One variation not shown here involves allowing the bridge to shut down the secondary bus while the primary bus is active.

The primary bus speed is equal to the secondary bus speed. The clock signal is passed through the bridge with only a slight skew. The **CLKRUN#** signal is basically passed through the bridge also. However, special care must be taken for the bi-directional nature of **CLKRUN#**. The following description explains the bi-directional controls for the buffers. Refer to Table 5 and Figure 13 for more information.

Table 5: CLKRUN# Terminology

Terms	Definitions
PCR#	Value of the **CLKRUN#** signal on the primary bus (that is, high or low).
SCR#	Value of the **CLKRUN#** signal on the secondary bus.
PEN	Enable for the open drain **CLKRUN#** driver on the primary bus. Note: The bridge can only drive the primary bus **CLKRUN#** low. Following the **CLKRUN#** protocol, the bridge drives the signal low for only two clock cycles, then releases it. The primary bus central resource takes over the low drive after that.
SEN	Enable for the s/t/s **CLKRUN#** driver on the secondary bus. Note: The bridge acts as the central resource for **CLKRUN#** on the secondary bus. It will drive **CLKRUN#** high for one cycle, then a keeper will hold it active. It will drive **CLKRUN#** low as indicated in the following description.

State A:

Upon power-up, the clock is running. Therefore, the primary bus central resource drives **CLKRUN#** low (PCR# = 0), and the bridge in turn drives **CLKRUN#** low on the secondary bus(SCR# = 0). PEN, the primary bus driver, is disabled. SEN, the secondary bus driver, is enabled.

State B:

The next action possible: The primary bus central resource attempts to stop/slow the clock by first driving **CLKRUN#** high (PCR# = 1). SEN remains enabled until the next clock. After that it disables its driver, enabling its keeper. PEN remains disabled.

Two things could happen next: either 1) a device on the primary bus pulls **CLKRUN#** low, or 2) a device on the secondary bus pulls **CLKRUN#** low.

State C:

SCR# = 0. PEN must become enabled for two clocks, then become disabled. SEN must also become enabled, disabling the keeper. SCR# = PCR# = 0.

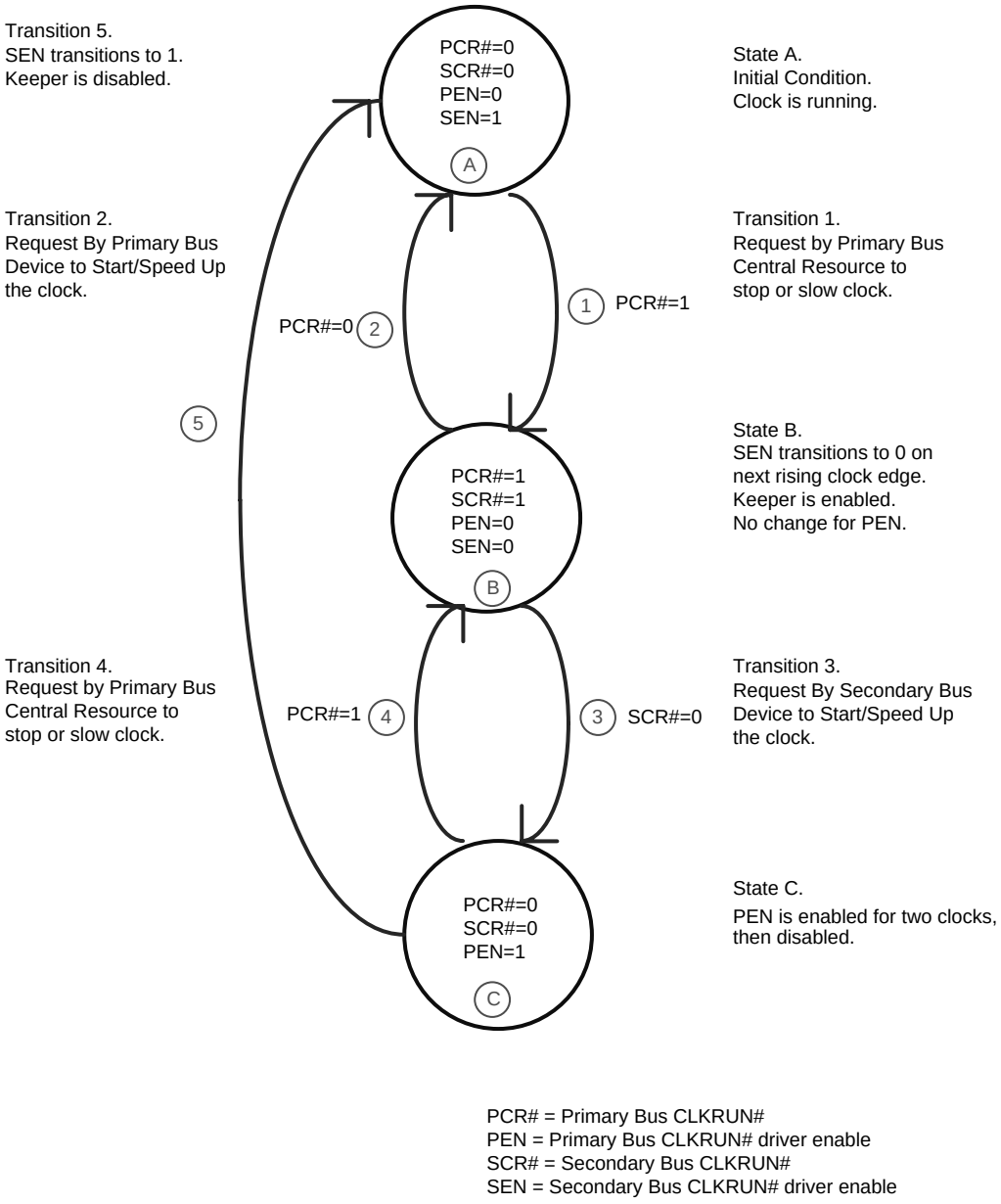

Figure 13: State Diagram for Case A: Skewed Clocks

6.2. Case B: Divided Clocks

When the clock for the secondary bus is derived from the primary bus clock, the bridge will have to act as "central resource" for the secondary bus, controlling **CLKRUN#** as well as ensuring clock run protocol timing is met.

6.2.1. Description of Flow Diagram: Divided Clocks

The flow starts with a condition of both the primary and secondary clocks running. See the flow diagram in Figure 14. Here, both PCR# and SCR# (primary and secondary **CLKRUN#** signals) are driven low.

The next condition to occur is a primary bus request to stop the clock. The primary bus central resource will drive PCR# high. The **CLKRUN#** protocol guarantees that there will be at least four additional primary clocks after PCR# is sampled high.

At this time, the bridge must notify the secondary bus devices of the request to stop the clock. It does so by driving SCR# high.

After SCR# is sampled high, the Secondary Clock Counter starts counting the number of secondary clocks. The bridge must ensure that at least four secondary clocks are given before the clock is allowed to be slowed/stopped on the secondary bus.

Any active cycles on the primary or secondary bus (**FRAME#** active) causes a return to all clocks running.

When the Secondary Clock Counter reaches four, the clock is allowed to slow/stop.

Prior to a secondary clock count of four, the bridge must ensure that the primary clock keeps running by asserting PCR# low whenever it is sampled high (this protocol is described in detail in Section 2.1.4.3. Maintaining the Clock. According to this protocol, the bridge must assert PCR# low within two primary clocks of it being sampled high to achieve an uninterrupted clock.

If there is a secondary request to not slow/stop the clock (by SCR# being asserted low) the flow returns to all clocks running.

Devices should not gate **CLKRUN#** with **FRAME#** to determine an active cycle. In this protocol, the clock resource may assert a stop/slow clock request on the same cycle a master starts a cycle. This will not cause an error condition since the clock is still running. The clock resource must withdraw its request to stop/slow the clock by driving **CLKRUN#** low.

See Figure 15, a timing diagram example of this flow. Here, the primary clock is divided by four to generate the secondary clock. When the primary clock is divided by four as shown, at least 23 primary clocks are needed before the secondary clock can be slowed/stopped.

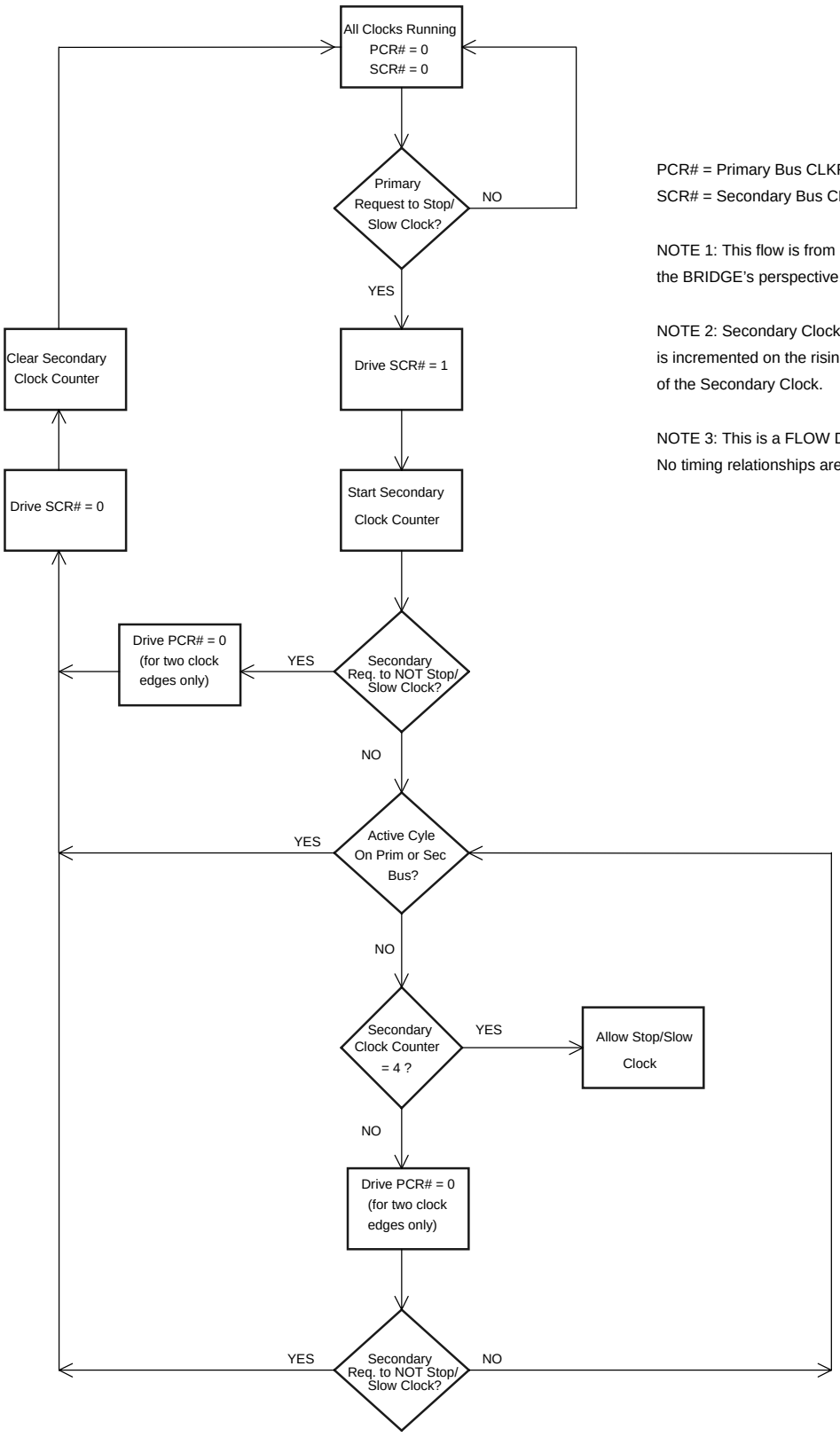

Figure 14: Flow Diagram for Case B: Divided Clocks

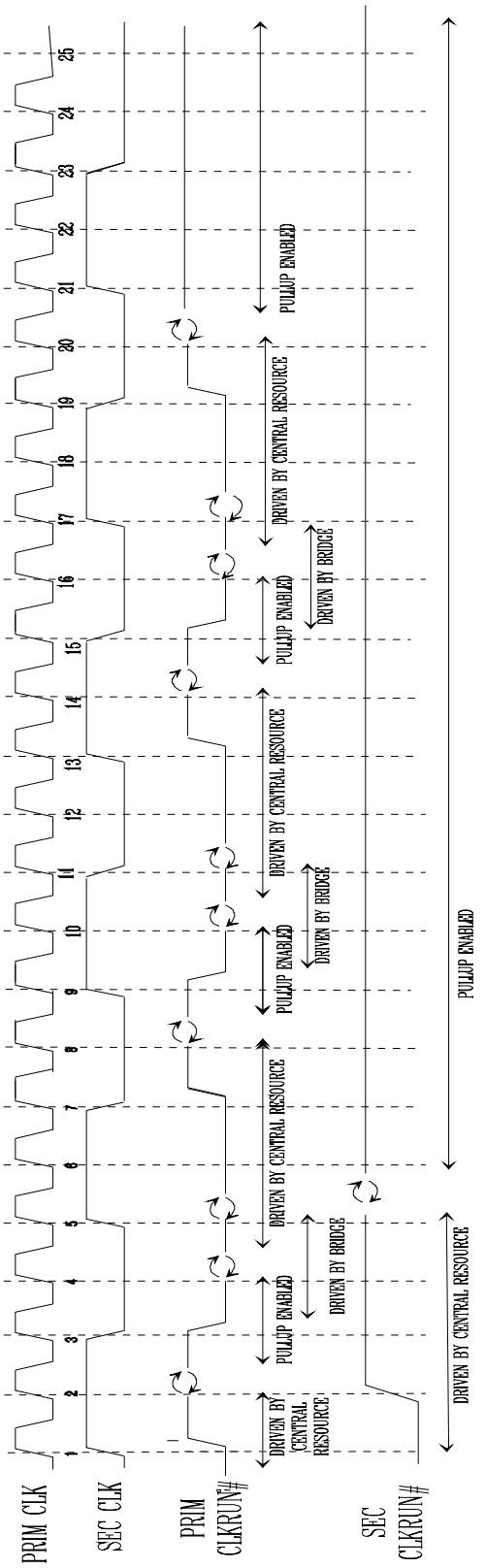

Figure 15: Timing Example for Case B: Divided Clocks

6.3. Case C: Asynchronous Clocks

When the buses are asynchronous to each other, the bridge designer has many options. Either bus can be stopped while the other is running full speed (as long as accesses do not need to cross the bridge). The bridge must initiate the clock run protocol to restart a stopped clock due to an active request on either bus. The bus states follow Figure 13.

6.4. Secondary Bus Clock Not Controlled By Bridge

When the clock control mechanism for the secondary bus is not handled by the bridge, the clock run protocol can still be used. **CLKRUN#** must be coupled (as always) with the clock source/controller.

One implementation involves an independent piece of hardware that receives the clock and redrives it to the bridge and each of the secondary bus devices. For this implementation, the bridge does not have control of the clock, therefore it cannot handle the **CLKRUN#** protocol. The independent piece of hardware must handle **CLKRUN#** as well.

Another implementation can have the clock distributed to each device from a central system resource. In this case, again the bridge does not have control of the clock and cannot handle the **CLKRUN#** protocol. The central resource must handle **CLKRUN#** - the central resource designer can make the choice whether to stop/slow the buses separately or together. The designer can also determine (rough estimate by the number of clocks he drives) the loading on **CLKRUN#**. If the loading is too large for one pin, he can supply two pins and simply OR them together in the central resource. There will be no contention of the signals since only the central resource can drive the signal high.

7. Use of Mixed Clock Run Support Components

Power management techniques vary from system to system, as does power usage. The Clock Run (**CLKRUN#**) signal is designed to introduce fine grain clock control, that is, maximum power savings, into mobile designs. With this signal, systems can stop or slow the interface clock between every active cycle, if they wish.

With the introduction of **CLKRUN#**, there will exist in the marketplace older components that do not support this signal. A system designer has several choices when confronted with a "power managed" design. He can pick only components for the motherboard that support **CLKRUN#** or he could mix components but not utilize the **CLKRUN#** protocol (he does the latter by tying **CLKRUN#** low on all components). Both of these choices are extremes, and tend to lead to disadvantages. For example, when picking only components with **CLKRUN#**, system design is restricted to new parts only. When the capability is available, not using the **CLKRUN#** protocol wastes functionality.

The system designer has another choice. He can architect his system both to utilize **CLKRUN#** for those parts that support this feature while still allowing the use of older components (in particular, those that are compatible with **CLKRUN#**). As was mentioned earlier, power management techniques vary from system to system, and an innovative system designer will tailor these techniques to utilize the varying support of **CLKRUN#** in his components.

7.1. Characteristics of Devices

Consider the component choices a designer has. Each type of device has features which allow or inhibit mobile design and **CLKRUN#** functionality and compatibility. For example, when attempting to architect a power managed design, active and static power consumption will eliminate some of the available component choices. It is not unreasonable to consider that non-static implementations will also be eliminated by some designers.

Motherboard Components: PCI motherboard components "may operate at any single fixed frequency up to 33 MHz, and may enforce a policy of no frequency changes" (Reference: *PCI Local Bus Specification*.). However, PCI Mobile components are recommended to have a static interface design, thereby supporting a frequency range down to DC. It is these static devices that are useful to a mobile designer and may be compatible with **CLKRUN#**.

Add-In Cards: PCI has add-in cards which "must work with any clock frequency between DC and 33 MHz." (Reference: *PCI Local Bus Specification*.). By this definition, add-in cards are by nature static designs and must work in a slow/stop clock environment. (Work is defined here as: not lose device data with variable frequency clocks. Understandably, buffer over/under run may be a problem).

CardBus: CardBus add-In cards support **CLKRUN#** if needed, and also support a frequency range down to DC.

Master: Master devices are strongly recommended to participate in the clock run protocol. Careful attention must be used with these devices since buffer over/under run may occur if the clock is stopped or slowed.

Slave: Slaves do not initiate accesses and therefore will not need to participate in the clock run protocol. A feature has been added to the clock run protocol to allow slaves to participate in the protocol, however, this function is more of an enhancement to the device and is not needed for proper functionality.

Once non-static implementations are eliminated, of all the devices listed here, master devices on the motherboard that do not support **CLKRUN#** are the devices which may malfunction in a **CLKRUN#** environment. The system designer can disable the clock run protocol when these master devices are enabled (if the central resource does not have an enable/disable feature for **CLKRUN#**, the pin can be tied low on the motherboard), or he can design his system to ensure that the master has access to the PCI bus when needed.

7.1.1. Bus Access Algorithm

When **CLKRUN#** is not supported by one or more master devices in the design, the system designer will need to devise an algorithm for roughly determining when the PCI Master needs access to the bus. Many pieces of information are available to do this, such as buffer sizes (motherboard components), bus latency rules (PCI masters must accept certain amounts of latency for access to the bus), and the Maximum Latency field (described below).

By using the PCI Max_Lat field, the system designer can determine the opportune times to stop/slow and restart the PCI clock. This field is used with other PCI fields to determine required bus bandwidth for the device and, therefore, its latency timer

programming. It is "used for specifying how often the device needs to gain access to the PCI bus. The value specifies a period of time in units of 1/4 microsecond. Value of zero indicates device has no major requirements for the settings of the latency timers." (Reference: *PCI Local Bus Specification*)

This field can also be used to roughly determine when a master will require a clock edge to generate a request. If the system wants to stop the clock, it would restart the clock periodically to service requests. If the system wants to slow the clock, it would need to guarantee that the clock frequency gave the device a rising edge often enough for it to produce a request to empty/fill its buffers. The device would need to supply edges twice as often as its Max_Lat value. For example, if the field value is 2 microseconds, a stopped clock system would restart the clock every microsecond and a slowed clock system would supply a rising edge every microsecond.

If more than one master in the system does not have **CLKRUN#** support, the smallest Max_Lat value would be used for the system.

Figure 16 shows the flow diagram a system designer would follow to determine system **CLKRUN#** functionality. Please note that determining **CLKRUN#** support for the motherboard can be determined at design time.

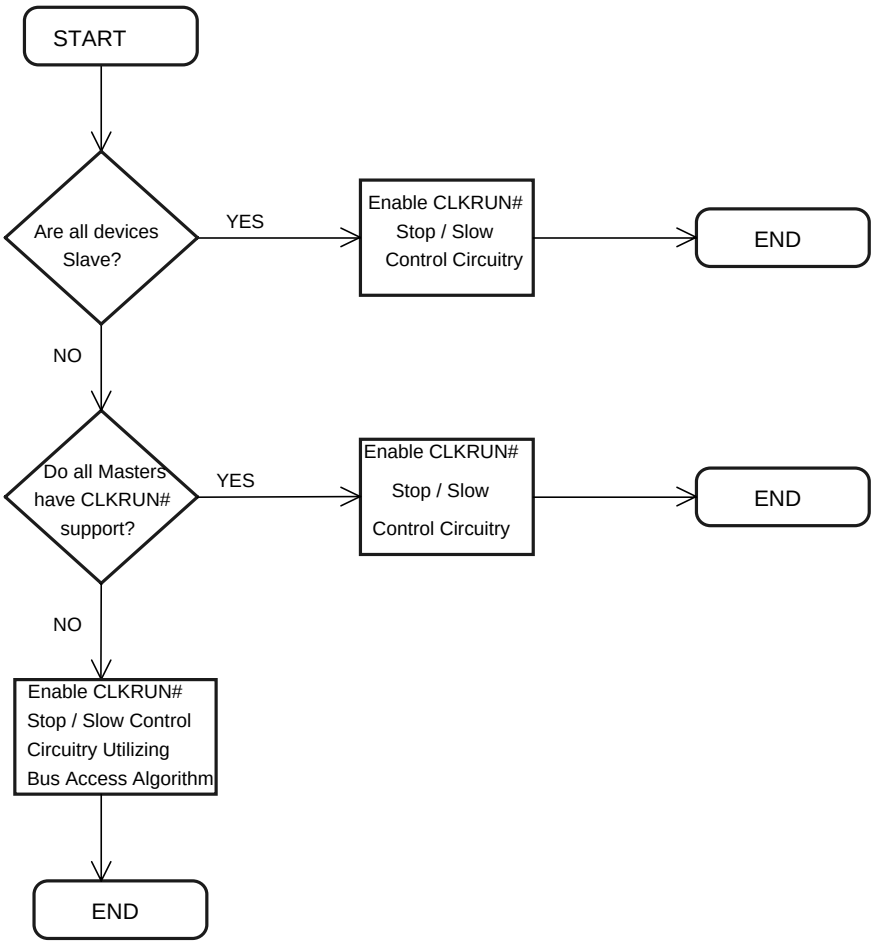

Figure 16: Determining Clock Run Support on the Motherboard

7.2. Clock Slowing/Stopping and Add-in Cards

Since add-in cards are required to operate from DC to 33 MHz, they are, therefore, static devices and work well in a power managed system. If the add-in card is a slave, it does not need to participate in the protocol and will function properly in a **CLKRUN#** environment.

If the add-in card is a master card, the system can use a bus access algorithm to periodically restart the clock.

See Figure 17 for determining at boot time clock slowing/stopping compatibility with add-in cards.

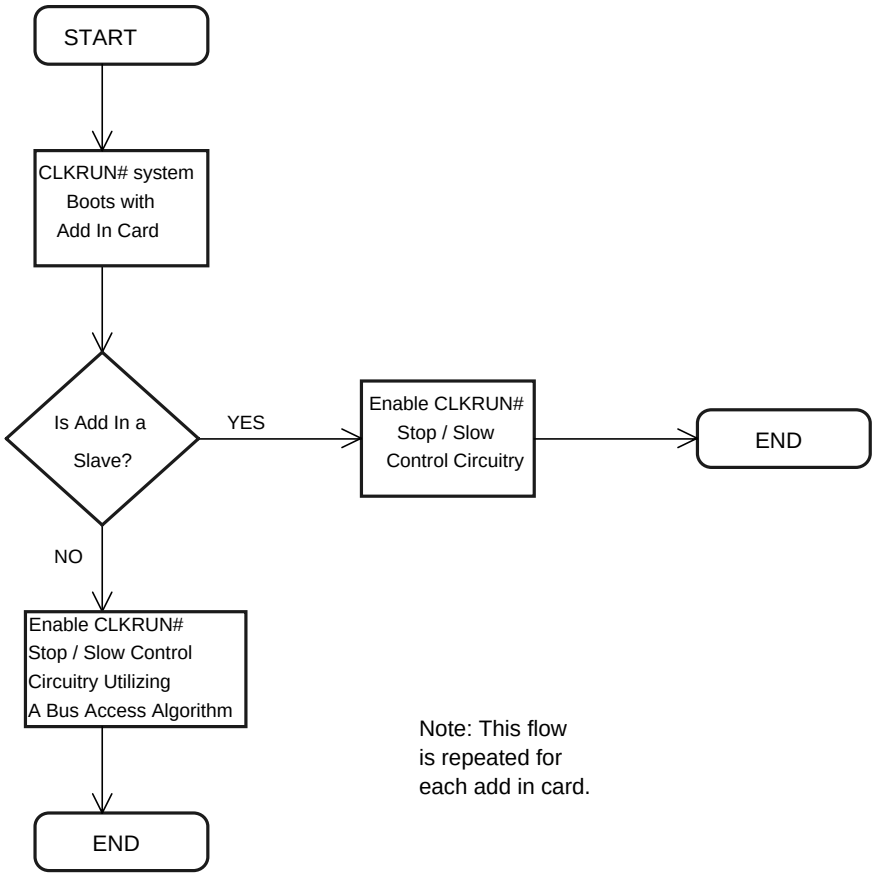

Figure 17: Determining Clock Slowing/Stopping Compatibility with an Add-in Card

8. PCI Buffer Leakage Control

In any power managed system it is desirable to turn off power to unused portions of circuitry. This reduces power to a lower level than the simpler method of stopping clocks. Unfortunately, CMOS devices have inherent to their structure parasitic clamp diodes to both supplies. Any high level signal input to a CMOS circuit will supply power to that circuit and prevent achieving the low power consumption required.

In PCI system designs, a typical scenario could include several PCI peripherals devices. Upon detection of a suspend state request, the system could desire to power down some but not all of the PCI devices. However, it is likely that some PCI devices may not be

powered down, perhaps a PMU or Central resource which must remain powered to monitor wake up sources. The system power management unit will typically power off the desired PCI devices and indicate the entry into a suspend state via some sideband mechanism to the remaining powered PCI devices. These powered PCI devices must then enter a leakage control state. The PCI bus would be inactive at this time.

In this case where not all PCI devices are powered down, care must be taken to choose which peripheral devices on the PCI interface must be separately powered down. If a device is not implemented for residing on an inactive PCI bus on which some devices are powered down, it will at best lose power to leakage and in the worst case partially power up powered down devices through their input structures. This latter case can induce CMOS latch up on power up and impair system reliability.

Each type of signal on the interface has a specific buffer characteristic (input, output, or other) which dictates device design implementation allowing for peripheral power down without leakage through PCI input and output structures. The PCI leakage control features should also prevent parasitic power up of powered down peripherals via PCI control signals.

System implementers should place a device's interface in a leakage control state directly upon entry of a power down suspend state. Each type of interface circuit must be considered separately when examining leakage control. The responsibilities for leakage control can be separated into PCI central resource and PCI agent functions. The PCI central resource is responsible for placing the bus into a leakage controlled state. This is not to say that the logic responsible for this function must physically reside with other central resource logic, but rather that it is a function that needs to be implemented once for a given PCI bus. PCI agents must logically disconnect from the bus.

8.1. Leakage Controlled System Example

The following sections show a potential implementation for a leakage controlled system. This example is shown here to illustrate the problem. Please note that other market solutions are available. Figure 18 is a system diagram showing a leakage scenario.

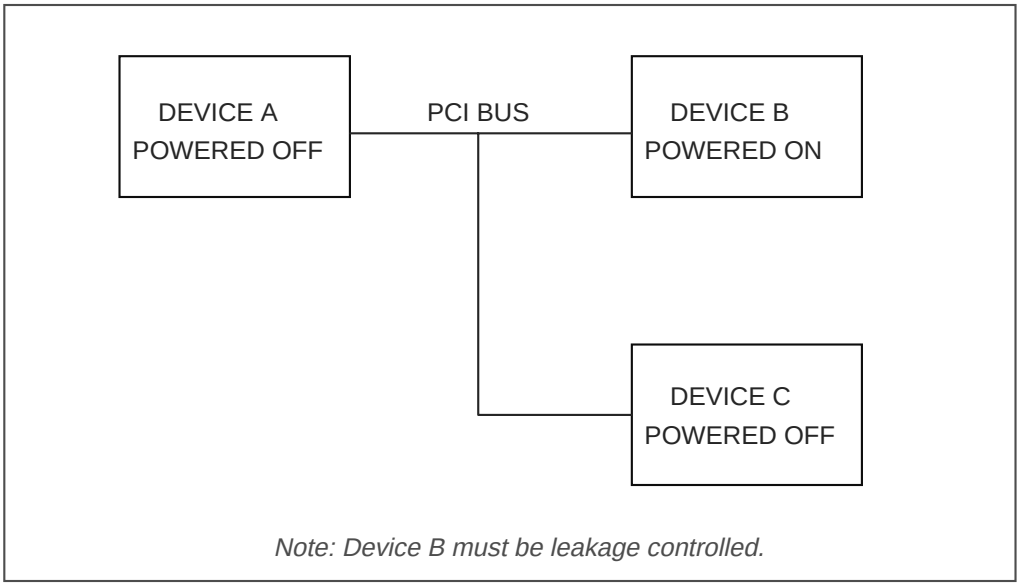

Figure 18: System Scenario for Leakage Problems

The following sections describe device specific mechanisms for solving leakage problems. First, the physical implications for each type of I/O circuit are discussed. Next, leakage states are given followed by a discussion of a mechanism for entering and exiting this leakage controlled state.

8.1.1. Input Circuits

In the leakage control state an input circuit should not supply any current to the external pin nor should it provide a path for cross over currents, without regard to the external voltage applied. Figure 19 is a diagram of a typical CMOS input buffer.

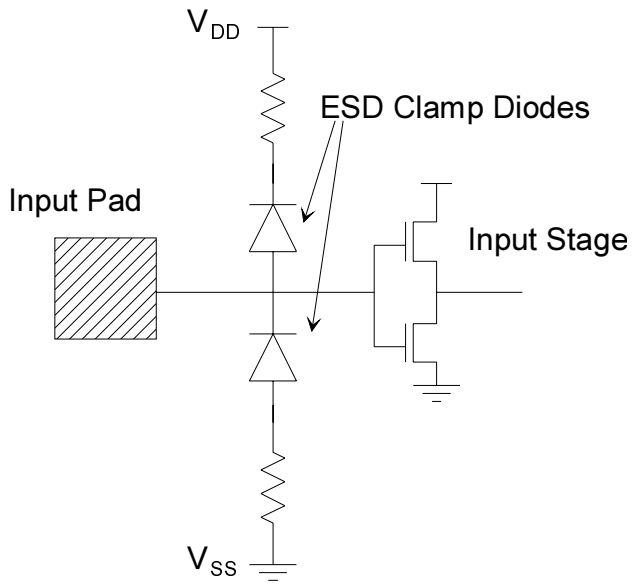

Figure 19: PCI Input Circuit

If both Vdd and Vss approach ground potential and the input pad is over a diode drop above ground, the ESD clamp diode to Vdd will be forward biased. This will cause the powered down device to be parasitically powered via the input pad.

Additionally, if Vdd is at operational levels and the input pad is not at a rail both transistors in the input stage will be on and shunting current. Care must be taken to prevent this.

The one way to solve both issues is for all inputs to be driven or pulled low, in this case the functionality associated with the input must be disabled during the leakage control state.

In summary, for this example, when in the leakage control state all powered input circuits could do the following:

- disable all pull-up or pull-down devices
- prevent any shunting currents through the input totem pole
- disable all functionality associated with the input

8.1.2. Output Circuits

In the leakage control state an output circuit should not supply any current to the external pin without regard to the external voltage applied. Figure 20 shows a generalized output circuit. Note that the ESD clamps are missing in the diagram. The typical structure of a standard output stage can usually supply the needed ESD protection without additional circuitry.

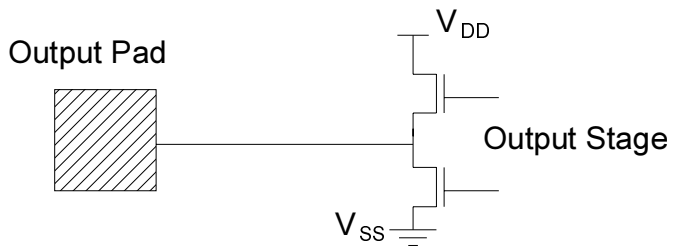

Figure 20: PCI Output Circuit

In the case of outputs, it is important to prevent the output stage from sourcing current unnecessarily. As mentioned before, this could happen if the output is driving a "high" level into an unpowered input stage.

Additionally, output stages may contain passive pull ups and pull downs which are also potential supplies for leakage current. When in a leakage control state, these level holding circuits should be disabled or otherwise accounted for.

Finally, the task of eliminating a source for un-powered input stage leakage falls on the powered output drivers. If a signal is allowed to float or is otherwise tri-stated the connected un-powered input stages will have forward biased ESD clamp diodes. This causes the floating signal to parasitically power the un-powered devices. This leads to excessive leakage current from the powered device to the un-powered device and increases the possibility of latch up when power is again applied to the shut down device. Therefore, a powered output stage must drive its signal level low for the duration of the leakage control state.

Again in summary, for this example, when in the leakage control state all powered output circuits could do the following:

- disable all pull-up devices
- force the output to a driven low state

The application of a driven low or “0” state is valid in this case because no external voltage source is attached to the circuit. Note, also, that the transition to the “0” state is independent of the active state of the signal pin. Peripheral circuitry must disable any critical functionality before switching power.

8.1.3. Input / Output Circuits

In the leakage control state, an input/output circuit does not supply any current to the external pin or provide a path for cross over currents. Figure 21 is a typical I/O circuit. Note that it is simply a combination of an input and an output stage.

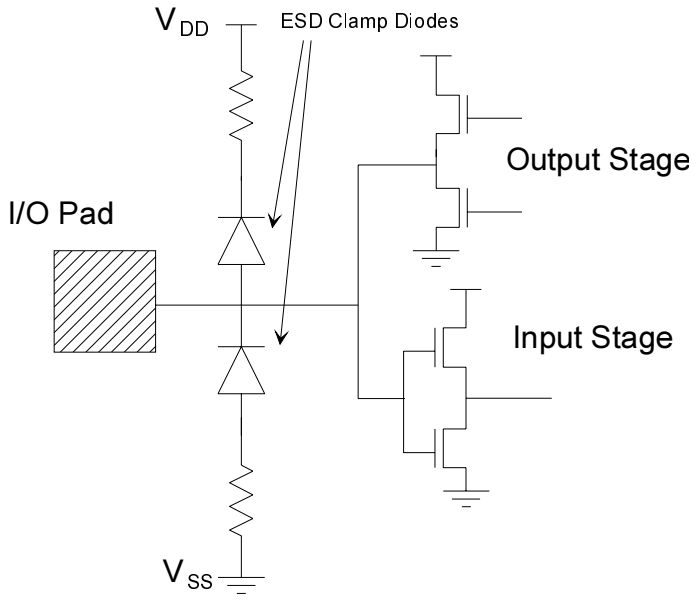

Figure 21: PCI I/O Circuit

Since I/O circuits are a combination of input and output stages the leakage control issues are simply a combination of input and output issues. The solutions are also similar with one exception.

Since I/O pads are frequently used in bused signal situations the output stages of the powered I/O pads should not be strongly driven low as they can be in a pure output case. The possibility of a multiply sourced low level must, instead, be dealt with by implementing a passive pull down. The use of a passive pull down prevents the possibility of excessive leakage currents if two or more powered I/O pads are connected and transition to the leakage control state and attempt to force the signal low at slightly different times.

When in the leakage control state all powered input/output circuits could do the following:

- disable all pull-up devices
- force all output drivers to three-state
- force the output low with a weak pull-down
- prevent any shunting currents through the input totem pole
- disable all functionality associated with the input

The application of a weak pull down only enabled in the leakage control state is used in this case because it is possible that an external device could be driving the signal.

8.1.4. Clock Inputs

Clock inputs are assumed to be always driven. When a clock is stopped, it must be stopped in the “0” state.

8.1.5. Clock Outputs

Clock outputs will always be driven. If the clock is stopped, the driven state must be “0”. During transitions into and out of the leakage control state clock outputs must not glitch.

8.1.6. PCI Side Band Signals

Typically, there are PCI bus side band signals which control pins on device interfaces. As a result, the PCI bus may be reduced in power by stopping the clock or be powered down by shutting off power to all attached devices.

The interface is broken into two parts:

- signals connected only to the side band interfaces
- signals connected only to the PCI bus

Signals connected to side band interfaces may follow the guidelines established here for the input, output, input/output signal types but are not addressed here specifically.

The following buffer state descriptions are defined for PCI peripherals. Note that a distinction is made between a driven low and resistive low assertion during the leakage state. If a device has the only driver on a signal, it is driven low. This is denoted by a "0" in the tables. If the signal is shared it must essentially be placed in three-state and pulled down. This is denoted by an "L" in the tables.

8.1.7. Core PCI Interface

For this example, signals connected only to the PCI bus can be driven by the central resource as shown in Table 6.

Table 6: PCI Bused Signals

Signal	Direction	Inactive State	Leakage State
RST#	Out	1	0
FRAME#	I/O	1	L
IRDY#	I/O	1	L
TRDY#	I/O	1	L
STOP#	I/O	1	L
LOCK#	I/O	1	L
SERR#	I/O	1	L
PERR#	I/O	1	L
DEVSEL#	I/O	1	L
C/BE#(3:0)	I/O	Prev. state	L
PAR	I/O	Prev. state	L
SBO#	I/O	1	L
SDONE	I/O	1	L

PCICLK	Out	0	0
AD(31:00)	I/O	Prev. state	L

Note that **RST#** should be treated with care. **RST#** can be taken low by the central resource to prevent leakage of the inactive signal to unpowered devices. However, the functional response of any powered agents to this active reset level while in the leakage control state is left to the implementer.

8.1.8. Bus Master Interfaces

REQ/GNT pairs normally connect to an individual peripheral and are powered off together. Table 7 lists the states of **REQ/GNT** pairs.

Table 7: REQ/GNT Pairs

Signal	Direction	Inactive State	Leakage State
REQn#	out	1	0
GNTn#	in	1	Three-State

8.1.9. Interrupt Interface

The interrupt interface consists of four interrupt lines which cannot be hard grouped with other peripheral interfaces. Each of these interrupt inputs may be powered off individually. Table 8 lists the PCI interrupts' states.

Table 8: PCI Interrupts

Signal	Direction	Inactive State	Leakage State
INTA#	out	1	L
INTB#	out	1	L
INTC#	out	1	L
INTD#	out	1	L

8.1.10. Mechanism for Entering / Exiting Leakage Controlled State

Thus far, the discussion of a specified mechanism for getting into and out of the leakage control state has been avoided. Instead, the focus has been on how to prevent leakage once some of the devices are powered down. It hasn't been stated that power up is done with reset asserted but clearly powered down devices will have to be reset. As for powered devices, once they are in a leakage control state they are logically disconnected from the PCI bus and therefore should ignore any oddball signals while the other devices are powering up. Implicit in this is some sort of sequencing which could go something like this:

1. PMU decides it is time to power down and asserts a shutdown signal. This shutdown signal puts powered devices into leakage control state.
2. The PMU then removes/grounds power to the devices to be powered-down.
3. The system is now in a suspend state with a minimum number of powered devices.
4. PMU decides it's time to resume and asserts PCI reset and powers up the system.

5. When power is good the PMU deasserts PCI reset and the leakage control signal.

There are probably as many permutations of this sequence and implementations thereof as there are designers of portable systems. The efficiency of this mechanism is likely to be a differentiating feature. Here, only the basic requirements to enable partial PCI power down are given.

A possible implementation of the mechanism for entrance/exit from the leakage control state is a sideband signal. Upon assertion of this signal, devices would enter a leakage controlled state. Upon deassertion of this signal, the devices would leave this state. For more information, see Figure 22.

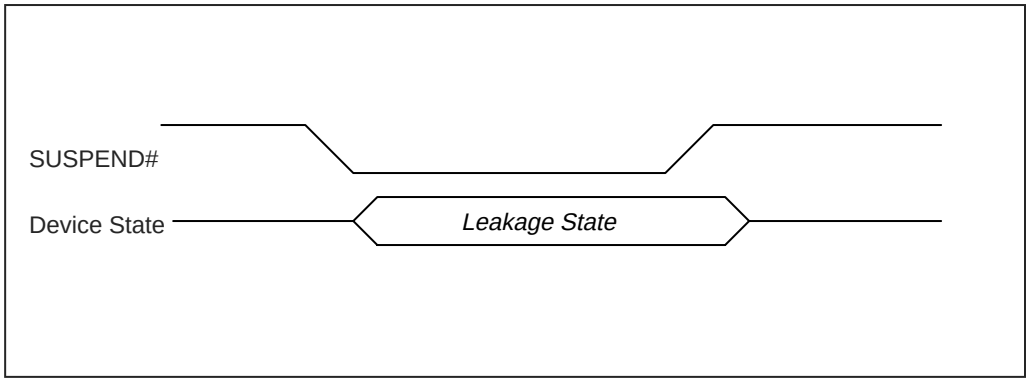

Figure 22: Example of a Leakage State Entrance / Exit Mechanism

Note that this mechanism cannot be implemented under software control. Upon entrance into a leakage controlled state, all communication across the PCI bus is shut down.