

UNITED STATES PATENT AND TRADEMARK OFFICE

---

BEFORE THE PATENT TRIAL AND APPEAL BOARD

---

BLACKBERRY CORP.,  
Petitioner,

v.

CYPRESS SEMICONDUCTOR CORP.,  
Patent Owner.

---

**DECLARATION OF ANDREW WOLFE PH.D.**  
in Support of Petition for *Inter Partes* Review of  
U.S. Patent No. 6,493,770

Mail Stop PATENT BOARD  
Patent Trial and Appeal Board  
U.S. Patent and Trademark Office  
P.O. Box 1450  
Alexandria, VA 22313-1450

## TABLE OF CONTENTS

I.	INTRODUCTION .....	1
II.	QUALIFICATIONS .....	1
III.	MATERIALS CONSIDERED AND PREPARED .....	7
IV.	SUMMARY OF OPINIONS .....	8
V.	LEGAL PRINCIPLES USED IN ANALYSIS .....	9
A.	Patent Claims in General .....	10
B.	Prior Art .....	11
C.	Unpatentability - Anticipation .....	12
D.	Unpatentability - Obviousness .....	13
VI.	BACKGROUND OF THE PATENT AND RELEVANT TECHNOLOGY .....	15
VII.	THE '770 PATENT .....	19
VIII.	CLAIM CONSTRUCTION .....	25
IX.	OVERVIEW OF THE PRIOR ART .....	27
A.	Patent Owner's Admitted Prior Art ("APA") .....	27
B.	U.S. Patent No. 6,073,193 to Yap ("Yap") .....	31
C.	U.S. Patent No. 5,628,028 to Michelson ("Michelson") .....	36
D.	PCCextend 100 User's Manual ("PCCextend") .....	38
E.	U.S. Patent No. 5,862,393 to Davis ("Davis") .....	40
X.	UNPATENTABILITY ANALYSIS .....	45
A.	The Claims of the '770 Patent .....	45

B.	Claim 11 is unpatentable under 35 U.S.C. § 102(b) as being anticipated by the USB Specification V1.0.....	48
C.	Claims 1, 5, 7, 10, 11, and 15–17 are unpatentable under 35 U.S.C. § 103(a) as being obvious over the APA in view of Yap .....	62
D.	Claims 2, 3, 12, and 13 are unpatentable under 35 U.S.C. § 103(a) as being obvious over APA in view of Yap and Michelson .....	80
E.	Claims 1–3, 10, 11–13, 16–18, and 20 are unpatentable under 35 U.S.C. § 103(a) as being obvious over Michelson in view of PCCextend and Davis .....	86
F.	Claims 5, 7, 15, 19 are unpatentable under 35 U.S.C. § 103(a) as being obvious over Michelson in view of PCCextend, Davis, and the APA .....	108
G.	Claims 18–20 are unpatentable under 35 U.S.C. § 102(e) as being anticipated by Yap.....	114
XI.	CONCLUDING STATEMENTS.....	118

I, Andrew Wolfe, hereby declare as follows:

## **I. INTRODUCTION**

1. I am currently a consultant at Wolfe Consulting.

2. I have been retained in this matter to provide various opinions regarding U.S. Patent No. 6,493,770 (the “’770 patent”). I am being compensated for my work in this matter at my ordinary hourly consulting rate. My compensation in no way depends upon the outcome of this proceeding.

3. I have been advised that Cypress Semiconductor Corp. owns the ’770 Patent. I have no financial interest in the ’770 patent.

## **II. QUALIFICATIONS**

4. I have more than 30 years of experience as a computer architect, computer system designer, personal computer graphics designer, educator, and as an executive in the electronics industry.

5. In 1985, I earned a B.S.E.E. degree in Electrical Engineering and Computer Science from The Johns Hopkins University. In 1987, I received an M.S. degree in Electrical and Computer Engineering from Carnegie Mellon University. In 1992, I received a Ph.D. in Computer Engineering from Carnegie Mellon University. My doctoral dissertation proposed a new approach for the architecture of a computer processor.

6. In 1983, I began designing touch sensors, microprocessor-based computer systems, and I/O (input/output) cards for personal computers as a senior design engineer for Touch Technology, Inc. During the course of my design projects with Touch technology, I designed I/O cards for PC-compatible computer systems, including the IBM PC-AT, to interface with interactive touch-based computer terminals that I designed for use in public information systems. I continued designing and developing related technology as a consultant to the Carroll Touch division of AMP, Inc., where in 1986 I designed one of the first custom touchscreen integrated circuits.

7. From 1986 through 1987, I designed and built a high-performance computer system as a student at Carnegie Mellon University. From 1986 through early 1988, I also developed the curriculum, and supervised the teaching laboratory, for processor design courses.

8. In the latter part of 1989, I worked as a senior design engineer for ESL-TRW Advanced Technology Division. While at ESL-TRW, I designed and built a bus interface and memory controller for a workstation-based computer system, and also worked on the design of a multiprocessor system.

9. At the end of 1989, I (along with some partners) reacquired the rights to the technology I had developed at Touch Technology and at AMP, and founded The Graphics Technology Company. Over the next seven years, as an officer and a

consultant for The Graphics Technology Company, I managed the company's engineering development activities and personally developed dozens of touchscreen sensors, controllers, and interactive touch-based computer systems.

10. I have consulted, formally and informally, for a number of fabless semiconductor companies. In particular, I have served on the technical advisory boards for two processor design companies: BOPS, Inc., where I chaired the board, and Siroyan Ltd., where I served in a similar role for three networking chip companies—Intellon, Inc., Comsilica, Inc, and Entridia, Inc.—and one 3D game accelerator company, Ageia, Inc.

11. I have also served as a technology advisor to Motorola and to several venture capital funds in the U.S. and Europe. Currently, I am a director of Turtle Beach Corporation, providing guidance in its development of premium audio peripheral devices for a variety of commercial electronic products.

12. From 1991 through 1997, I served on the Faculty of Princeton University as an Assistant Professor of Electrical Engineering. At Princeton, I taught undergraduate and graduate-level courses in Computer Architecture, Advanced Computer Architecture, Display Technology, and Microprocessor Systems, and conducted sponsored research in the area of computer systems and related topics. I was also a principal investigator for DOD research in video technology and a principal investigator for the New Jersey Center for Multimedia

Research. From 1999 through 2002, I taught the Computer Architecture course to both undergraduate and graduate students at Stanford University multiple times as a Consulting Professor. At Princeton, I received several teaching awards, both from students and from the School of Engineering. I have also taught advanced microprocessor architecture to industry professionals in IEEE and ACM sponsored seminars. I am currently a lecturer at Santa Clara University teaching graduate courses on Computer Organization and Architecture and undergraduate courses on electronics and embedded computing.

13. From 1997 through 2002, I held a variety of executive positions at a publicly-held fabless semiconductor company originally called S3, Inc. and later called Sonicblue Inc. For example, I held the positions of Chief Technology Officer, Vice President of Systems Integration Products, Senior Vice President of Business Development, and Director of Technology. At the time I joined S3, the company supplied graphics accelerators for more than 50% of the PCs sold in the United States.

14. Beginning in 1998, I began to work closely with S3's largest customer, Diamond Multimedia, to explore possible opportunities for a merger. My investigation included evaluating the technology, market, and business model related to the "Diamond Rio PMP300," the first commercially viable flash-memory MP3 player. In 1999, I led the merger negotiations between the two companies,

managed significant parts of company integration, and, after the merger was complete, worked on new product development. Soon after the merger with Diamond, we introduced the “Diamond Rio PMP500,” a portable music player that included Universal Serial Bus (“USB”) capability and also the ability to play downloaded files purchased from the Audible.com website. We also developed relationships with MP3 music vendors, including eMusic and MP3.com.

15. While at Diamond, we also developed the Rio 600 and 800 MP3 players, which included support for digital rights management (“DRM”) protected music using protocols from Microsoft. During the development of the PMP500 and the Rio 600, we also developed a music delivery platform and webstore backend service for selling DRM-protected music. In 1999, this business segment was spun out as a separate company called RioPort.com. I served on the RioPort.com board of directors and became involved in their product and technology strategy. I also managed engineering and marketing for the Rio product line for a period of time as an interim general manager.

16. I served as a board member and technical advisor at KBGear Inc. from 1999-2001. KBGear Inc. designed and produced digital cameras and music players that included USB ports and flash memory.

17. I have published more than 50 peer-reviewed papers in computer architecture and computer systems and IC design.



18. I also have chaired IEEE and ACM conferences in microarchitecture and integrated circuit design and served as an associate editor for IEEE and ACM journals.

19. I am a named inventor on 36 U.S. patents and 24 foreign patents.

20. In 2002, I was the invited keynote speaker at the ACM/IEEE International Symposium on Microarchitecture and at the International Conference on Multimedia. From 1990 through 2005, I have also been an invited speaker on various aspects of technology and the PC industry at numerous industry events including the Intel Developer's Forum, Microsoft Windows Hardware Engineering Conference, Microprocessor Forum, Embedded Systems Conference, Comdex, and Consumer Electronics Show, as well as at the Harvard Business School and the University of Illinois Law School. I have been interviewed on subjects related to computer graphics and video technology and the electronics industry by publications such as the Wall Street Journal, New York Times, Los Angeles Times, Time, Newsweek, Forbes, and Fortune as well as CNN, NPR, and the BBC. I have also spoken at dozens of universities including MIT, Stanford, University of Texas, Carnegie Mellon, UCLA, University of Michigan, Rice, and Duke.

21. Based on my technical education, and my years of professional experience as both an engineer and as an educator, I consider myself to be an expert in the field of computer architecture and computer system design, consumer

electronics, and computer programming, including computer busses, interfaces, and input/output ports. Moreover, I am very familiar with the operation and functional capabilities and limitations of commercial computers and computer peripherals existing during the late 1990s.

22. My professional experience with computer peripheral device interface design and with USB technology, as well as my educational background, is summarized in more detail in my C.V., which is included as Exhibit 1021 to the petition.

### **III. MATERIALS CONSIDERED AND PREPARED**

23. In forming the opinions expressed below, I considered the '770 patent and the other patents in its family (U.S. Patent Nos. 6,012,103 and 6,249,825) (collectively the "USB Patents") and their file histories as well as the prior art references and related documentation discussed herein. I have also relied upon my education, background, and experience.

24. In addition, I have reviewed the declaration of Geert Knapen that was presented with respect to a prior IPR petition related to the '770 patent. In most cases, I found the presentation of pertinent facts and the accompanying analysis in that declaration to be both accurate and well written. Furthermore, in many cases, my relevant opinions are identical to Mr. Knapen's. In these cases, I have duplicated Mr. Knapen's language in this declaration to simplify the presentation to

the PTAB. Where my opinions differ from Mr. Knapen's or I felt that a different form of presentation is preferable, I have written new text.

#### **IV. SUMMARY OF OPINIONS**

25. Based on my investigation and analysis, and for the reasons set forth below, it is my opinion that all of the elements and steps recited in claims 1–3, 5, 7, 10–13, and 15–20 of the '770 patent are disclosed in prior art references and that those claims are anticipated and/or rendered obvious in view of these references. In particular, I have relied primarily on the six prior art references identified below in support of my opinions:

- (1) Patent Owner's Admitted Prior Art ("APA") (Ex. 1001);
- (2) U.S. Patent No. 6,073,193 to Yap ("Yap") (Ex. 1002);
- (3) U.S. Patent No. 5,628,928 to Michelson ("Michelson") (Ex. 1003);
- (4) PCCextend100 User's Manual ("PCCextend") (Ex. 1004);
- (5) U.S. Patent No. 5,862,393 to Davis ("Davis") (Ex. 1005);
- (6) Univeral Serial Bus Specification v1.0, January 15, 1996, Copyright 1996, Compaq Computer Corporation, Digital Equipment Corporation, IBM PC Company, Intel Corporation, Microsoft Corporation, NEC, Northern Telecom ("USB 1.0 Specification") (Ex. 1013);

26. Besides the above documents, I have also considered the following references in preparing my declaration:

- (1) Prosecution History of U.S. Patent 6,012,103 (Ex. 1006);
- (2) Prosecution History of U.S. Patent 6,249,825 (Ex. 1007);
- (3) Prosecution History of U.S. Patent 6,493,770 (Ex. 1008);
- (4) U.S. Patent No. 5,590,273 to Balbinot (Ex. 1014)
- (5) U.S. Patent No. 6,338,109 to Snyder (Ex. 1015)
- (6) Quinnell, Richard A., “USB: A Neat Package with a Few Loose Ends,”

EDN Magazine (October 24, 1996) (Ex. 1016).

(7) Levine, Larry. PCMCIA Primer, pp. 117-130 (M&T Books 1995)  
(Ex. 1017).

(8) PCMCIA PC Card Standard Release 2.01, pp. 3-2 to 3-5; 4-2 to 4-7;  
4-10 to 4-19; 4-28 to 4-31; 4-34 to 4-37; 5-2 to 5-5; 5-12 to 5-21; 5-23; 5-48 to 5- 51;  
6-6 to 6-17 (published 1992) (Ex. 1018)

(9) PCMCIA Card Services Specification Release 2.0, pp. 3-2 to 3-7; 3- 14  
to 3-17; 3-20 to 3-25; 3-28 to 3-29; 5-78 to 5-79 (published 1992) (Ex. 1019)

(10) U.S. Patent No. 5,537,654 to Bedingfield (Ex. 1020)

27. The bases for my opinions are set forth in greater detail below and in  
the claim charts attached as Appendix A.

## **V. LEGAL PRINCIPLES USED IN ANALYSIS**

28. I am not a patent attorney and I am presenting no opinions on the law  
related to patent validity. BlackBerry’s attorneys have explained certain legal

principles to me that I have relied on in forming my opinions set forth in this declaration.

29. I was informed that my assessment and determination of whether or not claims 1–3, 5, 7, 10–13, and 15–20 of the '770 patent are unpatentable must be undertaken from the perspective of what would have been known or understood by someone of ordinary skill in the art as of the earliest priority filing date of the USB Patents—July 2, 1997. From analyzing the USB Patents and the relevant prior art, it is my opinion that a person of ordinary skill in the relevant art for the '770 patent (“PHOSITA”) would be sufficiently skilled in the design of peripheral devices used in connection with computer systems to understand and practice the prior art discussed in this declaration. Unless otherwise specified, when I state that something would be known to or understood by one skilled in the art or possessing ordinary skill in the art, I am referring to someone with this level of knowledge and understanding.

**A. Patent Claims in General**

30. I have been informed that patent claims are the numbered sentences at the end of each patent. I have been informed that the claims are important because the words of the claims define what a patent covers. I have also been informed that the figures and text in the rest of the patent provide a description and/or examples

and help explain the scope of the claims, but that the claims define the breadth of the patent's coverage.

31. I have also been informed that an "independent claim" expressly sets forth all of the elements that must be met in order for something to be covered by that claim. I have also been informed that a "dependent claim" does not itself recite all of the elements of the claim but refers to another claim for some of its elements. In this way, the claim "depends" on another claim and incorporates all of the elements of the claim(s) from which it depends. I also have been informed that dependent claims add additional elements. I have been informed that, to determine all the elements of a dependent claim, it is necessary to look at the recitations of the dependent claim and any other claim(s) on which it depends.

32. I have also been informed that patent claims may be expressed as "methods" or "apparatuses/devices/systems." That is, I have been informed that a patent may claim the steps of a "method," such as a particular way to perform a process in a series of ordered steps, or may claim a combination of various elements in an "apparatus," "device," or "system."

## **B. Prior Art**

33. I have been informed that the law provides categories of information (known as "prior art") that may anticipate or render obvious patent claims. I have been informed that, to be prior art with respect to a particular patent in this

proceeding, a reference must have been published, or patented, or be the subject of a patent application by another, before the priority date of the patent. I have also been informed that a person of ordinary skill in the art is presumed to have knowledge of all prior art. I have been asked to presume that the reference materials that I opine on, *i.e.*, the APA; U.S. Patent No. 6,073,193 to Yap; U.S. Patent No. 5,628,028 to Michelson; PCCextend 100 User's Manual; U.S. Patent No. 5,862,393 to Davis; and USB 1.0 Specification, are prior art from a technical perspective – that is, all were available to a person of ordinary skill in the art on or before the priority date of the patent.

### **C. Unpatentability - Anticipation**

34. I have been informed and understand that determination of whether a patent claim is “anticipated” is a two-step process. First, the language of the claim is construed as it would be understood by one of ordinary skill in the art at the time of the filing of the patent application. Reference is made to the intrinsic evidence of record, which includes the language of the claim itself and other issued claims, the patent specification, and the prosecution history. Words in a claim will be given their ordinary or accustomed meaning unless it appears that the inventor used them differently. The prosecution history may limit the interpretation of the claim, especially if the applicant disavowed or disclaimed any coverage in order to obtain allowance of the claim.

35. Second, I understand that after the patent claim has been construed, determining anticipation of the patent claim requires a comparison of the properly construed claim language to the prior art on an element-by-element basis.

36. I understand that a claimed invention is “anticipated” if each and every element of the claim has been disclosed in a single prior art reference, or has been embodied in a single prior art device or practice, either explicitly or inherently (i.e., necessarily present or implied).

37. I understand that although anticipation cannot be established by combining references, additional references may be used to interpret the anticipating reference by, for example, indicating what the anticipating reference would have meant to one having ordinary skill in the art.

**D. Unpatentability - Obviousness**

38. I have been informed that, even if every element of a claim is not found explicitly or implicitly in a single prior art reference, the claim may still be unpatentable if the differences between the claimed elements and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person of ordinary skill in the art. That is, the invention may be obvious to a person having ordinary skill in the art when seen in light of one or more prior art references. I have been informed that a patent is obvious when it is only a combination of old and known elements, with no change in their respective



functions, and that these familiar elements are combined according to known methods to obtain predictable results. I have been informed that the following four factors are considered when determining whether a patent claim is obvious: (1) the scope and content of the prior art; (2) the differences between the prior art and the claim; (3) the level of ordinary skill in the art; and (4) secondary considerations tending to prove obviousness or nonobviousness. I have also been informed that the courts have established a collection of secondary factors of nonobviousness, which include: unexpected, surprising, or unusual results; prior art that teaches away from the alleged invention; substantially superior results; synergistic results; long-standing need; commercial success; and copying by others. I have also been informed that there must be a connection, or nexus, between these secondary factors and the scope of the claim language.

39. I have also been informed that some examples of rationales that may support a conclusion of obviousness include:

- a) Combining prior art elements according to known methods to yield predictable results;
- b) Simply substituting one known element for another to obtain predictable results;
- c) Using known techniques to improve similar devices (or product) in the same way (e.g. obvious design choices);

- d) Applying a known technique to a known device (or product) ready for improvement to yield predictable results;
- e) Choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success-in other words, whether something is “obvious to try”;
- f) Using work in one field of endeavor to prompt variations of that work for use in either the same field or a different one based on design incentives or other market forces if the variations are predictable to one of ordinary skill in the art; and
- g) Arriving at a claimed invention as a result of some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior art reference or to combine prior art reference teachings.

40. I have also been informed that other rationales to support a conclusion of obviousness may be relied upon, for instance, that common sense (where substantiated) may be a reason to combine or modify prior art to achieve the claimed invention.

## **VI. BACKGROUND OF THE PATENT AND RELEVANT TECHNOLOGY**

41. The '770 patent relates to a system and method for interfacing a computer system to a peripheral device. A wide variety of peripheral devices were

common at the time of the '770 patent's priority date, examples of which included a computer mouse, keyboard, printer, network adapter, modem, data storage device, and computer monitor. Often these peripherals, particularly a network adapter, modem, or data storage device, were in the form of a PC card (also referred to as a PCMCIA card). Various specifications have been developed to facilitate interaction between a computer and a peripheral device. These specifications have included the Personal Computer Memory Card International Association (PCMCIA) Specification and the Universal Serial Bus (USB) Specification.

42. In the Background of the '770 Patent ("Background"), the patentee admits that it was known to connect a peripheral device to a computer using a USB connection. Ex. 1001, 1:50–2:4; 4:15–34; Fig. 1. The patentee also admits in the Background that, when the USB connector of a peripheral is inserted into a powered-up host computer or inserted into a powered-down host computer which is then powered up, the host computer detects the peripheral device and a configuration process known as "enumeration" begins which causes the peripheral device to be recognized by the host computer's operating system. *Id.* at 1:66–2:19.<sup>1</sup>

---

<sup>1</sup> The USB 1.0 Specification actually explained that enumeration is an ongoing activity for the bus and that it is only done at startup time for some busses. "4.6.3 Bus Enumeration Bus enumeration is the activity that identifies and addresses devices attached to a bus. For many buses, this is done at startup time and the

43. The Background further alleges that the only opportunity for associating a software device driver with a peripheral device is at the time when the enumeration process occurs. Ex. 1001, 2:20–28. “Thus, to alter the configuration or personality of a peripheral device, such as downloading new code or configuration information into the memory of the peripheral device, the host computer system must detect a peripheral device connection or a disconnection and then a reconnection.” *Id.* at 2:24–28.

44. This was admitted to be one of the “problems of known systems and methods. . . .” *Id.* at 2:37–40. Accordingly, it was admitted to be known that a peripheral device could have a first configuration and that a second configuration could be downloaded into the peripheral device over a computer bus. All of these features are also found in one or more of the prior art references discussed herein.

45. The Background describes that the problem that the host computer system must detect a physical disconnection and reconnection is solved by a switch which is connected to one of the USB data lines D+ and D-. Ex. 1001, 6:59–65;

---

information collected is static. Since the USB allows USB devices to attach to or detach from the USB at any time, bus enumeration for this bus is an on-going activity. Additionally, bus enumeration for the USB also includes detection and processing of removals.” Ex. 1013 at 32. “enumerating the USB is an on-going activity” Ex. 1013 at 31.

7:9–30. It was known that a host detects the connection of a peripheral device by monitoring voltage levels on one of the two USB data lines. *Id.* at 6:28–31; Ex. 1013 at 114. Thus, by changing the state of the data lines, the switch is “electronically simulating a physical disconnection or reconnection of the peripheral device,” as recited in independent claims 1, 11, and 18. However, as discussed in more detail below, it was well known in the prior art (*e.g.*, in U.S. Patent No. 6,073,193 to Yap; PCCextend 100 User’s Manual, and U.S. Patent No. 5,862,393 to Davis) to position a switch in the lines of a bus between a peripheral device and host computer which can be opened and closed to simulate a physical disconnection and reconnection and cause reconfiguration. The method of resetting a USB port after configuration was also well known. Ex. 1013 at 115–117, 119, 14, 29, 165–169, 221–222, 263. Also, the USB specification explained that certain devices had “hardware support for reset and suspend/resume signaling.” Ex. 1013 at 35. A reset on such a device sets the port state to “Disconnected.” Ex. 1013 at 223. Coming out of reset, attached devices are redetected. “Upon coming out of reset, a hub must detect which downstream ports have devices connected to them.” Ex. 1013 at 224. A reset that switches power on and off to simulate a disconnect was also part of the USB specification. Ex. 1013 at 132, 242. In fact, the USB specification discloses the existence of non-removable devices that can only be reset using this simulation process. Ex. 1013 at 264. Thus, the problem that a host

needs to detect a disconnection and reconnection to cause reconfiguration had a well-known solution in the prior art.

## **VII. THE '770 PATENT**

46. The background admits that physically disconnecting and reconnecting a peripheral device to reconfigure the peripheral device was known at the time of the invention. *See supra*, Section VI. This physical disconnection and reconnection caused a host computer to perform an enumeration process to recognize the requirements and capabilities of the device and select an appropriate device driver with which to use the peripheral device. *See, e.g.*, Ex. 1001, 1:66–2:18.

47. The '770 Patent relates to using an electronic circuit to simulate the disconnection and reconnection to take the place of an actual physical disconnection and reconnection. *Id.* at 2:62–3:8; 5:36–43.

48. Figure 2 of the '770 Patent (reproduced below) illustrates a USB system “in accordance with the invention.” Ex. 1001, 3:52–53; 4:64–65. The USB system includes a host computer with an operating system that stores “one or more peripheral device drivers, such as a first peripheral device driver 68” and a “plurality of different configuration information sets 70.” *Id.* at 5:2–13.

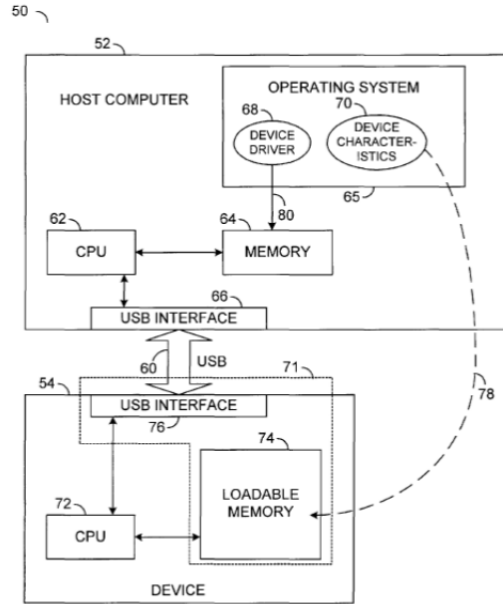


FIG. 2

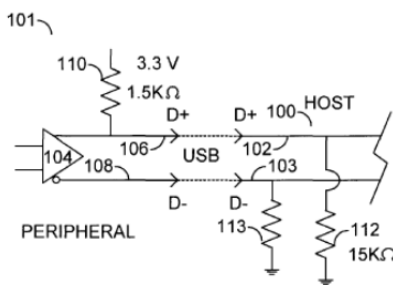
49. The host computer selects one of the plurality of configuration information sets, such as an updated configuration information set, to download to the peripheral device. Ex. 1001, 5:36–54. Instead of relying on a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device based on the updated configuration information set, the host uses an “electronic disconnect and reconnect method in accordance with the invention.” *Id.* at 5: 36–43. In other words, the “disconnect/connect cycle may be electrically simulated” so that “a change in the configuration information for a particular peripheral device may be implemented.” *Id.* at 2:66–3:1.

50. For example, the peripheral device may have a first configuration that is an “initial factory configuration of the peripheral device.” *Id.* at 5:44–48. “[W]hen the peripheral device is first connected to the USB, the configuration

information 70” for a second configuration, including “any microprocessor code applicable to the peripheral device and the appropriate configuration data for the peripheral device may be downloaded over the USB into the memory 74 of the peripheral device 54 as shown by the dashed arrow 78.” *Id.* at 5:48–54.

51. Then, the “electrical simulation of the disconnection and reconnection of the peripheral device . . . may be initiated and a re-enumeration process may occur.” *Id.* at 5:54–57. “During the re-enumeration process, the newly downloaded configuration information may be used to reconfigure the USB for the peripheral device and the host computer may select the appropriate software device driver 68 for the peripheral device based on the configuration information and load the device driver into memory 64 as shown by arrow 80.” *Id.* at 5:57–63.

52. According to the ’770 Patent, a conventional host computer USB interface circuit monitors the two USB data leads, labeled D+ and D-, to detect a disconnection and reconnection. *Id.* at 3:54-55; 6:17–44; Fig. 3.

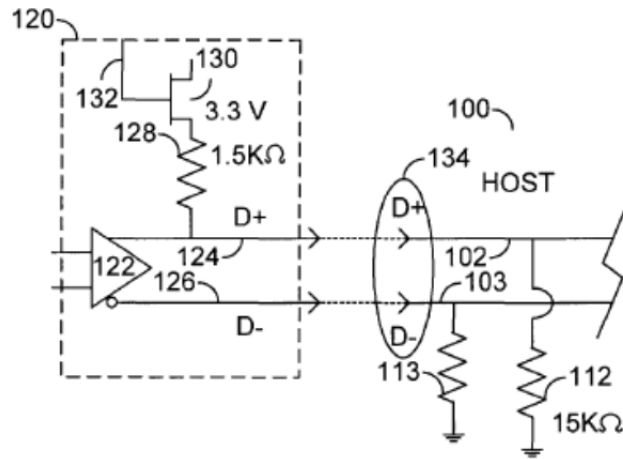


**FIG. 3**  
(PRIOR ART)



53. As shown in Fig. 3, when the host device and the peripheral device are connected, 3.3 V from a power bus is supplied to the D+ line. *Id.* at 6:17–31. “In operation, the host computer detects the connection of a peripheral device by monitoring the voltage levels of one of the two USB data leads.” *Id.* at 6:28–31. When the peripheral device is physically disconnected from the host computer, the connection from the 3.3 V supply voltage to the D+ line is broken as well, causing the host to measure zero volts on the D+ line. *Id.* at 6:31–36. Based on this measurement, the host computer “determines that no peripheral device is connected to the USB port.” *Id.* When that peripheral device or another peripheral device is connected to the host computer, “the 1.5 kΩ resistor 110 connected to a supply voltage of the peripheral device USB interface 101 adds a voltage to the D+ line and the D+ line at the host computer is pulled to above 3 volts which is detected as a connected peripheral device by the host computer and the host computer begins the enumeration process.” *Id.* at 6:36–44.

54. The '770 Patent describes simulating the disconnection/reconnection cycle by using a switch to break the connection between a supply voltage and the D+ line. *Id.* at 7:10–34; Fig. 4 (reproduced below).



**FIG. 4**

55. The switch 130 “may be a semiconductor switch such as a field effect transistor (FET),” and “may have a control lead 132 which may control the operation of the electrical switch.” *Id.* at 6:61–67. By opening the switch, “the D+ data lead is no longer connected to the supply voltage and the host computer determines that the peripheral device has been disconnected even though the peripheral device is still physically connected to the USB.” *Id.* at 7:13–18. “Similarly, when the electrical switch is closed again, the D+ data lead is again connected to the supply voltage and the host computer will detect that the peripheral device has been reconnected to the USB.” *Id.* at 7:18–21.

56. According to the ’770 Patent, the “electronic disconnection and reconnection of the peripheral device, as described above, in combination with the storage of the configuration information sets on the host computer permits the

configuration of the peripheral devices to be changed easily without requiring the physical disconnection and reconnection of a peripheral device.” *Id.* at 7:25–29.

57. According to the ’770 Patent, the USB interface system and method may be a single semiconductor chip, which may be incorporated into a plurality of peripheral devices. *Id.* at 3:12–15. “The chip may initially have a generic configuration (e.g., not specific to a particular peripheral device).” *Id.* at 3:15–17. “Then, the appropriate configuration information for a particular peripheral device and manufacturer may be downloaded to the chip, an electronic simulation of the disconnection and reconnection of the peripheral device occurs, the peripheral device is recognized as a new, manufacturer specific peripheral device and the appropriate software device driver is loaded into the memory of the host computer.” *Id.* at 3:17–24.

58. “For example, a plurality of different peripheral devices manufactured by different companies may each include a USB interface system in accordance with the invention.” *Id.* at 5:63–66. “The USB interface system for each peripheral device is identical (e.g., has a USB interface circuit and a memory) except that each memory may contain an identification code that is unique to, for example, a particular manufacturer.” *Id.* at 5:66–6:3. “When one of the peripheral devices is connected to the USB and the host computer, the appropriate configuration information for the peripheral device, based on the identification code, is

downloaded over the USB to the memory of the peripheral device and the appropriate software device driver is loaded into the memory of the host computer.”

*Id.* at 6:6–7:9.

59. According to the '770 Patent, one advantage of the electrical disconnection and reconnection, is that “since the peripheral device is physically connected to the bus during the electrical simulation, the peripheral device may utilize the electrical power supplied by the bus to operate the peripheral device.”

*Id.* at 3:1–3:6; see also 9:17–22 (“[B]ecause the peripheral device is not physically disconnected from the host computer, the peripheral device may use the electrical power available over the USB bus . . .”).

## **VIII. CLAIM CONSTRUCTION**

60. I have been informed that claim terms in the present proceeding are to be given their broadest reasonable interpretation in light of the specification in which it appears. Therefore, it is my understanding that each claim term of the asserted patent are to be given their broadest reasonable interpretation, as understood by one of ordinary skill in the art and consistent with the disclosure of the asserted patent.

61. Below, I set forth what I believe to be the broadest reasonable interpretation of certain claim terms in view of the specification, as well as the factual basis for those opinions. As to the other terms that I do not address in this

section, I used and applied the broadest reasonable interpretation of those terms in view of the specification as understood by one of ordinary skill in the art.

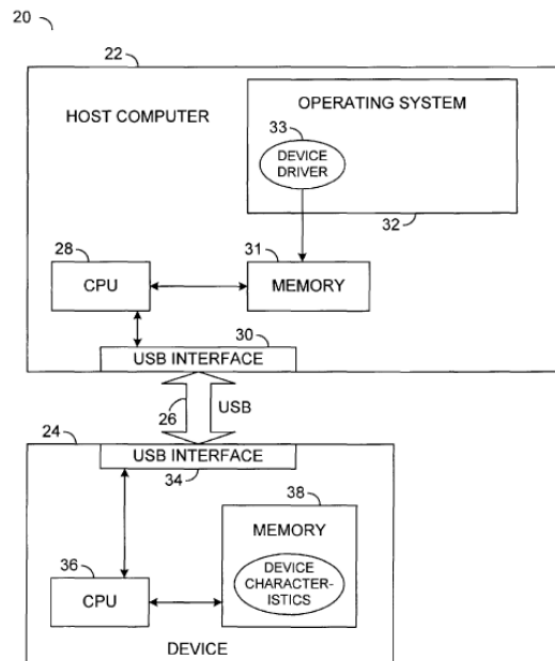
62. **“electronically simulate/simulating a physical disconnection and reconnection of the peripheral device”**: using an electronic circuit to perform an action, such as an electronic reset, associated with physical disconnection and reconnection of a peripheral device. This interpretation is the broadest reasonable interpretation that is consistent with the claims of the ’770 Patent and the rest of the specification. *See* Ex. 1001, 3:25–35; claims 1, 10, 11, 17, and 18. For example, independent claim 1 recites a second circuit configured to electronically simulate a physical disconnection and reconnection of a peripheral device, and dependent claim 10 recites “wherein said second circuit comprises a reset circuit configured to reset the first or second configuration of the peripheral device.” Similarly, independent claim 11 recites (B) electronically simulating a physical disconnection and reconnection of a peripheral device, and dependent claim 17 recites “wherein step (B) comprises electronically resetting the configuration of the peripheral device, controllable by the peripheral device.” Thus, the interpretation of the “electronically simulating” language must be broad enough so as not to exclude the reset circuit and resetting operation in the dependent claims. The interpretation of the “electronically simulating” language proposed herein encompasses the claimed reset circuit and resetting operation in the dependent claims, as well as the other

aspects of electronically simulating (such as simulating with a switch) in the claims, as well as the other aspects of electronically simulating (such as simulating with a switch) described in the patent (*see, e.g.*, 7:9–22), and is therefore the broadest reasonable interpretation consistent with the claims.

## IX. OVERVIEW OF THE PRIOR ART

### A. Patent Owner’s Admitted Prior Art (“APA”)

63. The Background describes the known Universal Serial Bus (USB). Ex. 1001, 1:50-2:29. Figure 1 of the ’770 patent is labeled Prior Art and is described as illustrating a standardized USB interface wherein a peripheral device 24 is connected to a host computer system 22 by a USB. Ex. 1001, 4:15–34; Fig. 1 (reproduced below).



**FIG. 1**  
(PRIOR ART)

64. According to the '770 Patent, it was known that when the peripheral device is initially connected to the USB, the host detects the peripheral device and an enumeration process begins in which the host determines the characteristics of the peripheral device by receiving configuration information from the memory 38 within the peripheral device and configures the USB according to the characteristics of the peripheral device. Ex. 1001, 1:66–2:4; 4:35–41. It was also known that the memory 38 for storing configuration information may be an erasable programmable read only memory (EPROM). Ex. 1001, 4:32–34. The Background further discloses that new code or configuration information could be downloaded from the host into the memory of the peripheral device over the USB. Ex. 1001, 2:24–29. These teachings are consistent with my understanding of the prior art at the time of the invention. *See, e.g.*, Snyder (Ex. 1015), 3:64–4:10 (reciting “loading a set of microprocessor instructions into a memory device coupled to a microprocessor of the microcontroller, the instructions loaded from an external computer”); 9:59–67 (“The storage medium can include, but is not limited to, any type of disk including floppy disks, optical discs, CD-ROMs, and magneto-optical disks, ROMs, RAMs, EPROMS, EEPROMS, magnetic or optical cards, or any type of media suitable for storing electronic instructions.”).

65. Further, according to the '770 Patent, once the enumeration process was complete in prior art USB systems, the CPU of the host computer could load an

appropriate software device driver for the peripheral device and the software applications executed by the host computer could communicate with the peripheral device using the USB. Ex. 1001, 4:51–56. Further, according to the '770 Patent, it was known that another software device driver could be loaded after a disconnection event and connection event. Ex. 1001, 4:56–62.

66. The Background describes that in a USB system, the only opportunity for associating software device drivers with a peripheral device is at the time when the peripheral device is plugged into the USB and the enumeration process occurs.<sup>2</sup> Ex. 1001, 2: 20–23. “Thus, to alter the configuration or personality of a peripheral device, such as downloading new code or configuration information into the memory of the peripheral device, the host computer system must detect a peripheral device connection or a disconnection and then a reconnection.” Ex. 1001, 2:24–28. According to the '770 Patent, the invention is directed to “a system and method for interfacing to a universal serial bus which avoids these and other problems of known systems and methods.” Ex. 1001, 2:36–40.

---

<sup>2</sup> As previously noted, the USB 1.0 Specification actually explains that enumeration is an ongoing activity for the bus and that it is only done at startup time for some busses. Enumeration was known to be repeated after a “reset,” which is a standard USB signal defined in the USB 1.0 Specification that does not require unplugging and plugging back in a USB device.



67. Based on the above passages, it is my opinion that the Background and the description of the prior art figures of the '770 Patent (Fig. 1, described in 4:15–62 and Fig. 3) (“Admitted Prior Art” or “APA”) admit that at least the following features were known in the prior art: (1) detecting a peripheral device connected to a computer bus; (2) a peripheral device that has a first configuration; (3) downloading information for a second configuration from the host computer into the peripheral device over the computer bus; (4) reconfiguring a peripheral device connected by a computer bus and port to a host computer; (5) physically disconnecting and reconnecting a peripheral device to reconfigure the peripheral device to a second configuration based on downloaded information for the second configuration; (6) a USB connector; (7) a USB peripheral device interface.

68. The APA is consistent with my own recollection and experience in the field at the time and of other prior art that I am aware of, including, *e.g.*, the prior art identified in this declaration. Moreover, in my opinion, the totality of the circumstances indicates that the Applicant considered the detecting and downloading features to be in the prior art. For example, in the prosecution history of the '825 and '770 Patents, the Applicant did not challenge the PTO's characterization of the detecting and downloading features as being Admitted Prior Art. Ex. 1007, pp. 53–54 and 62–63; Ex. 1008, pp. 70–71 and 90–91. When the Examiner relied on “Applicant's Admitted Prior Art” to teach the detecting and

downloading features of the claims, Applicant did not challenge this feature, and instead argued that the feature of electronically simulating a physical disconnection and reconnection was not taught in the secondary references. Ex. 1007, pp. 62–63; Ex. 1008, pp. 90–91.

69. Thus, it is my opinion that the features described above are merely characterizing what was already known in the prior art.

**B. U.S. Patent No. 6,073,193 to Yap (“Yap”)**

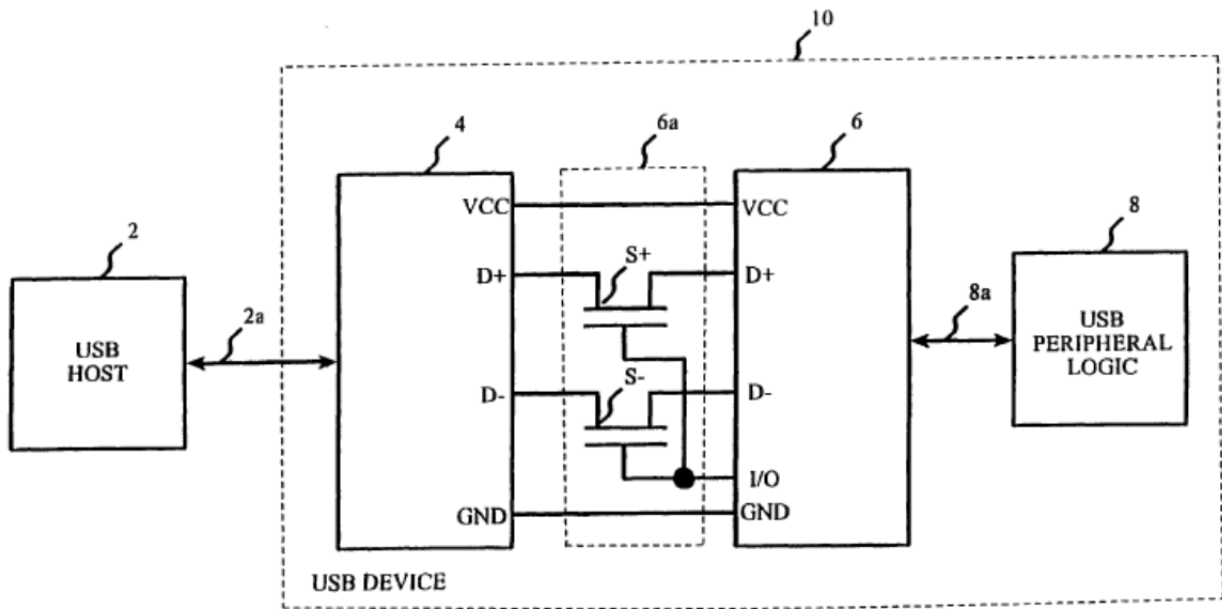
70. Yap teaches a method and apparatus for re-initializing a USB peripheral device when there is a USB microcontroller busy condition by disconnecting at least one data line of the USB microcontroller from a USB bus coupled to the USB microcontroller. Ex. 1002, at Abstract; 2:29–37.

71. Yap teaches a circuit that is configured to electronically simulate the physical disconnection and reconnection of the peripheral device. *Id.* at 3:60–4:10; 4:24–38; Figs. 2 & 3. Yap describes that a malfunction may occur in a USB device, wherein after the device is configured, the host may terminate the function of the USB device. Ex. 1002, 1:43–54. “When this occurs, (1) the user may have to re-boot the USB device or physically disconnect and then re-connect the USB device to allow the host computer to recognize and then re-configure the USB device....” Ex. 1002, 1:58–2:3. Yap appreciates that this “method defeats the

whole purpose of plug-and-play technology,” where devices are automatically configured by the host computer. *Id.*

72. Yap discloses that objects of the invention are to recover from a USB brown out condition “without a need to re-boot the USB device or physically disconnect and then re-connect the USB device.” *Id.* at 2:20–24. Thus, Yap expressly discloses that the disadvantage of having to physically disconnect and reconnect the USB device to allow the host computer to recognize and reconfigure the USB device is that it may be inconvenient. Yap describes a method and apparatus for recovering from a malfunction “without a need to re-boot the USB device or physically disconnect and then re-connect the USB device.” Ex. 1002, 2:22–24.

73. Figure 2 (reproduced below) of Yap shows a first embodiment where switching devices S+ and S-, shown as FET transistors, are coupled to the USB data lines D+ and D-. Opening and closing the lines “emulates the disconnect and re-connect procedure as specified in the USB specification v1.0, page 116.” Ex. 1002, 4:21–23. The switching devices are also described in Yap as transistors, *Id.* at 3:60–4:23, which are a type of solid state switching device.

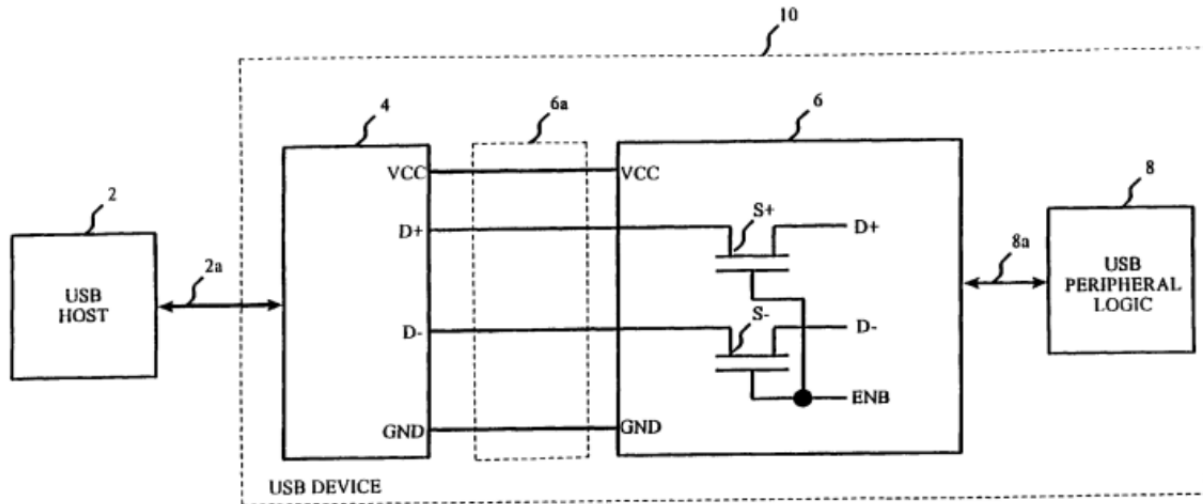


*Fig. 2*

74. Yap teaches that “[b]y disconnecting the D+ and D- data lines via switching devices S+ and S-, a physical removal of the USB device 10 may be simulated in order to allow the USB host to re-configure the USB device . . .” *Id.* at 4:6–10.

75. Yap teaches that opening the switching devices S+ and S- for a duration greater than 2.5 microseconds and then reconnecting them again simulates the disconnection and re-connection. Ex. 1002, 4:16–23. This duplicates the explanation in the USB specification. Ex. 1013 at 116.

76. Fig. 3 of Yap discloses an embodiment in which the switching devices S+ and S- are located in the microcontroller of the USB peripheral device. Ex. 1002, 4:24–38; Fig. 3 (reproduced below):



*Fig. 3*

77. A person of ordinary skill in the art would understand that in each of Yap's embodiments, one of the data lines, D+ or D-, must be pulled high through a 1.5KΩ resistor or its equivalent. This is a standard USB requirement. Full-speed devices, the most commonly available type at the time, would pull up the D+ line to a 3-3.6V level using a 1.5KΩ resistor or its equivalent. On the host side, D+ and D- are both connected to ground via a 15KΩ resistor. I have illustrated that understanding below. Ex. 1013 at 114.

78. Figure 7.5 and 7.6 from the USB 1.0 specification, reproduced below, illustrate this requirement. Referring to those figures, the left hand boxes of each figure show resistor elements R1 as rectangles connecting D+ and D- to a ground voltage, show as the small triangles.

The USB is terminated at the hub and function ends as shown in Figure 7-5 and Figure 7-6. Full speed and low speed devices are differentiated by the position of the pull-up resistor on the downstream end of the cable. Full speed devices are terminated as shown in Figure 7-5 with the pull-up on the D+ line. Low speed devices are terminated as shown in Figure 7-6 with the pull-up on the D- line.

The pull-up terminator is a  $1.5\text{ k}\Omega \pm 5\%$  resistor tied to a voltage source between 3.0 V and 3.6 V referenced to the local ground. The pulldown terminators are resistors of  $15\text{ k}\Omega \pm 5\%$  connected to their local ground.

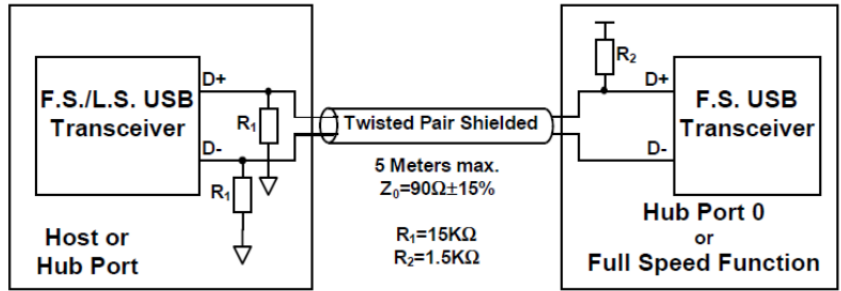


Figure 7-5. Full Speed Device Cable and Resistor Connections

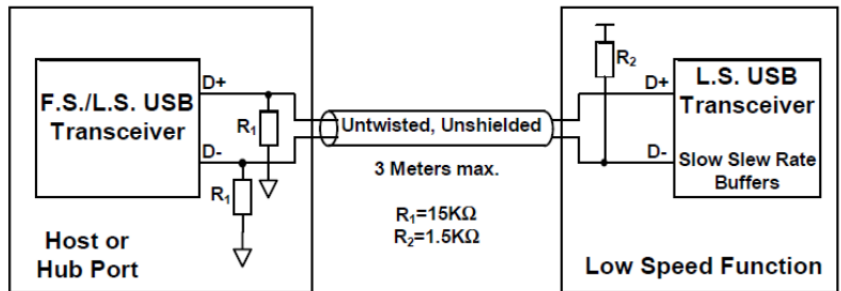
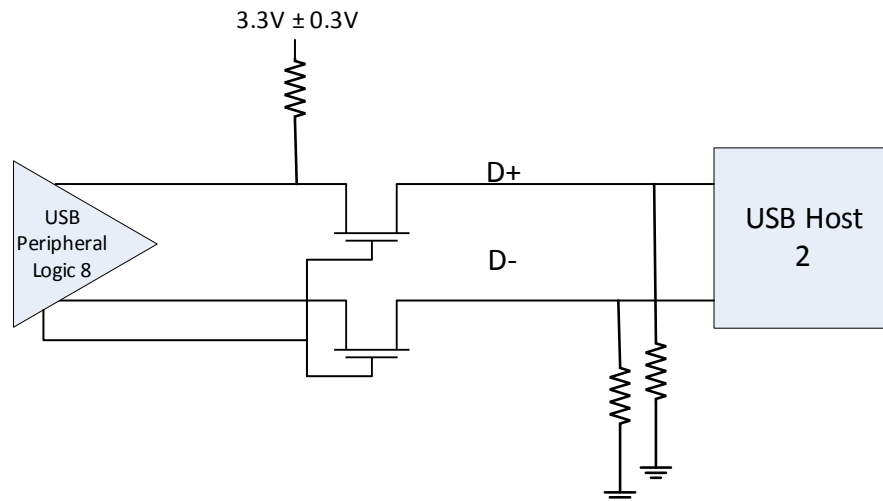


Figure 7-6. Low Speed Device Cable and Resistor Connections



Yap Figure 2 as understood by a PHOSITA in view of the USB specifications

79. I have prepared the illustration immediately above to show, in view of the USB 1.0 specification, how one of ordinary skill in the art would understand Figure 2 of Yap in its connection with a USB host.

**C. U.S. Patent No. 5,628,028 to Michelson (“Michelson”)**

80. Michelson relates to programming and reprogramming the hardware configuration of a PCMCIA card. Ex. 1003, 1:7–16. Michelson states that a “typical PCMCIA card includes a standard PCMCIA connector connected to a PCMCIA interface circuit through a standard PCMCIA bus.” Ex. 1003, 1:28–30. “[T]he host computer includes a PCMCIA adapter circuit coupled to a PCMCIA host socket which is mechanically and electrically connected to a PCMCIA card connector on the PCMCIA card” (*Id.* at 2:17–20), and “PCMCIA card 14 card connector 28 is inserted in PCMCIA host socket 18 of host computer 12” (*Id.* at 3:34–36). “Through a standard PCMCIA bus (i.e., PCMCIA address lines 62, data lines 64, and control lines 66) connected to PCMCIA connector 28”, the peripheral PCMCIA device receives data from the host computer. *Id.* at 4:13–23.

81. Michelson describes that when a PCMCIA card is inserted into the host socket of a host computer, an adapter in the host recognizes the insertion. Ex. 1003, 3:34–37. The processor in the host reads data in a Card Information Structure (CIS)

memory (such as an EEPROM) on the PCMCIA card and configures the host and the PCMCIA card to operate together. Ex. 1003, 2: 22–27. The CIS data sufficiently identifies the PCMCIA card to the host to enable the host computer and the PCMCIA card to operate together and to enable the processor to select the appropriate application software from the host memory. Ex. 1003, 3:34–49.

82. The CIS data specifically identify the card manufacturer and card identification (ID) number and includes a variety of set-up information. Ex. 1003, 3:49–54.

83. After an initial configuration, the processor 22 then executes the application software that corresponds to the PCMCIA card. Ex. 1003, 3:59–61. The application software causes the processor to either select a default field programmable gate array (FPGA) programming data file from host memory that corresponds to a particular application for the PCMCIA card or request input from the user as to which FPGA programming data file is to be selected from host memory. Ex. 1003, 3:61–66.

84. The host then downloads the data from the selected FPGA programming data file through PCMCIA adapter and bus to the PCMCIA interface chip, which then programs the FPGA by loading the data from the FPGA programming data file into the FPGA. Ex. 1003, 3:66–4:17. The interface chip

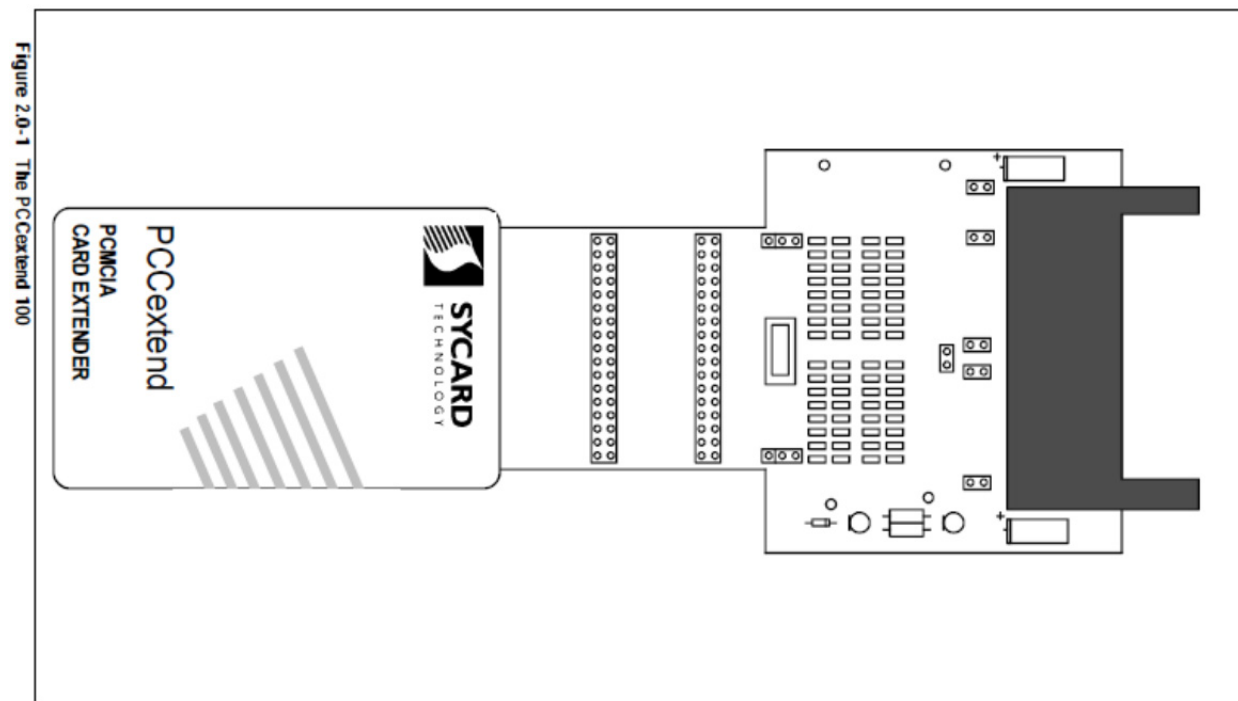


initiates reprogramming by a circuit, which sends a “reset” and “reprogram” signal.  
Ex. 1003, 4:17–22.

85. Michelson also discloses that the FPGA programming data files can be supplied with the PCMCIA card or new, additional, or updated FPGA programming data files can be obtained at a later time. Ex. 1003, 6:61–63.

**D. PCCextend 100 User’s Manual (“PCCextend”)**

86. PCCextend describes a PCMCIA extender card that simulates a card removal/insertion cycle. Ex. 1004, p. 1. The extender card is inserted into a desired slot in a host system. Ex. 1004, p. 1. A PC Card (i.e., a PCMCIA card) under test is inserted into the card connector of the extender card. Ex. 1004, Fig. 2.0-1 (reproduced below) shows the extender card described in PCCextend.



87. The “extender card is a debug tool for PCMCIA development and test.” Ex. 1004, p. 1. The extender card has test points and a termination and prototype area to allow access to all PC Card signals and to allow the user to add components to any signal. Ex. 1004, pp. 3–4.

88. PCCextend teaches that “[i]nsertion and removal of the extender and PC card should be done with care. The PC Card’s fragile connectors may be broken or bent if improper force is used. Both card and extender should be inserted straight without any lateral movement or force.” (Italics and bold omitted.) Ex. 1004, p. 1.

89. PCCextend describes that the extender card has a PCCswitch SW1, where “the PCCswitch can interrupt the card detect signals (-CD 1 and -CD2) to simulate a card removal/insertion cycle.” Ex. 1004, p. 3; see also schematic of extender card’s host side connector on p. 16 (reproduced below).



“connected to a computer 8 via a socket 14.” *Id.* at 4:16–23; 6:15–20; Fig. 1. A pair of card detect lines are used by a device controller to detect the card connected to the computer. *Id.* at 6:17–56; Fig. 1. Davis teaches that, in such devices, a device insertion signal is normally generated in response to inserting a device into a socket. Ex. 1005, 2:56–58. Davis further provides an inventive solution “by taking advantage of the known characteristics of device removal and insertion signals.” Ex. 1005, 2:66–3:3. More specifically, Davis teaches a first circuit (e.g., the device controller) that uses card detect lines 16a, 16b to detect the peripheral device connected to the port. For instance, Davis recites:

Turning now to FIG. 1, a device 12, such as a peripheral board or “card”, is connected to a computer 8 via a socket 14. **Once connected, a pair of card detect lines 16a and 16b connect the device 12 to a device controller 18.** In addition, a ground path 19 extends between the ground potential of the device 12 and to the card controller 18. Pull-up resistors 22a and 22b are located at the controller-side of the card detect lines 16a and 16b. Each pull-up resistor 22a and 22b is connected between a power source (Vcc) and a card detect line, thereby placing a logical high level on the card detect line when a device 12 is not connected to the socket 14. Although the pull-up resistors 22a and 22b are shown in FIG. 1 as discrete resistive components, those skilled in the art will appreciate that the pull-up resistors can be implemented as devices internal to the device controller 18. **A ground potential is located at the device side of the card detect liner<sup>3</sup> 16a and**

---

<sup>3</sup> One of ordinary skill in the art, reading the specification, would have recognized

**16b, thereby placing a logical low level on both card detect lines when the device 12 is properly connected to the socket 14. In response to connecting the device 12 to the socket 14, the logical high-level signal present on each card detect line 16a and 16b transitions to a logical low level.**

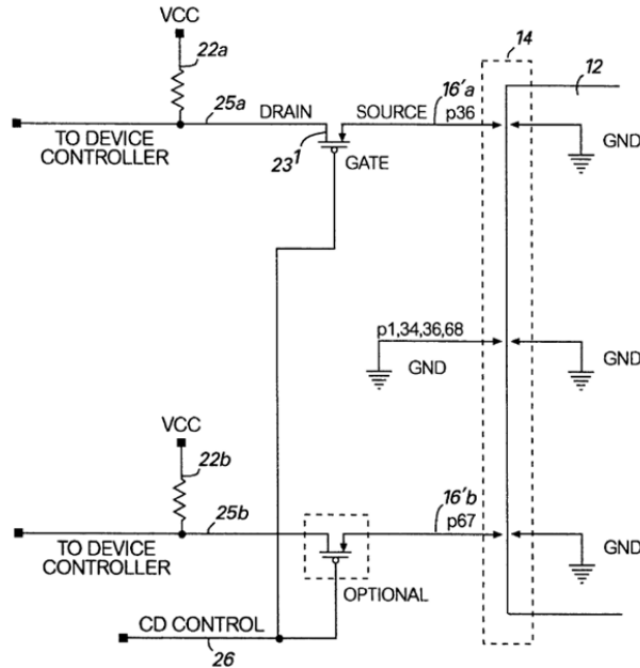
*Id.* at 6:14–54 (emphasis added).

93. Davis discloses an electronic switch in the card detect line that electronically simulates a physical disconnection and reconnection of a peripheral device. More specifically, Davis describes using FET switches to create an “‘apparent’ device removal event” and an “‘apparent’ device insertion event.” Ex. 1005, 3:32, 43. Fig. 3 (reproduced below) shows the FET transistors, which are configured to “break[] a signal path between the card detect lines 16a’ and 16b’ and system advisory lines 25a and 25b.” *Id.* at 7:23–35. Davis teaches that by “interrupt[ing] the passage of signals from the card detect lines 16a’ and 16b’ to the device controller 18,” the FET switches “trick[s] the device controller to take actions in response to the apparent removal of the device 12.” *Id.* at 9:45–52. “Significantly, the device 12 remains inserted within the socket 14, thereby leading to the presence of logical lower<sup>4</sup> levels signals on the card detect lines 16a’ and 16b’ that represent a device insertion event.” *Id.*; *see also* 9:18–32.

---

that “liner” contains a typographical error and should read “lines.”

<sup>4</sup> One of ordinary skill in the art, reading the specification, would have recognized



**FIG. 3**

94. One event “ ‘tricks’ [a] controller 18 into making a determination that the device 12 has been removed from the socket 14” and another event “ ‘tricks’ the controller 18 into making a determination that the device 12 has been inserted into the socket 14.” Ex. 1005, 11:15, 47.

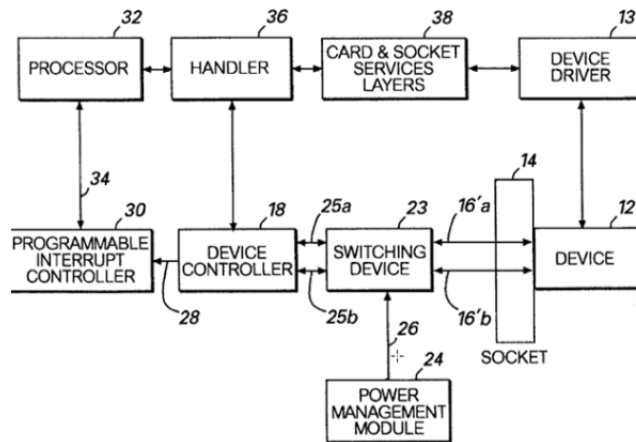
95. According to Davis, a sequence of power-down and power-up events can cause a computer device to enter a default state or a random state based on the loss of configuration information. Ex. 1005, 1:43–52. If the power management event is not communicated to the device driver to cause it to supply configuration information to the device, the only way to return a device that has lost its device

---

that “lower” contains a typographical error and should read “low.”

configuration to a useful state is to restart or re-boot the computer system. Ex. 1005, 1:43–49; 2:21–32. Davis presents a technique that uses the device removal and insertion signals normally generated by the physical removal or insertion of a device to advise a driver for a device about a power management event and cause configuration information to be sent to the device. Ex. 1005, 2:48–65.

96. Davis describes a switching device 23 connected in the card detect lines 16a' and 16b' between the peripheral device 12 (via the socket 14) and the device controller 18 in the host computer as shown in Fig. 2 (reproduced below):



**FIG.2**

97. Davis describes that “the switching device 23 can be implemented by an electronic switch, typically a field effect transistor (FET) or a bipolar transistor.” Ex. 1005, 7:31–34. The switches may be located in the card detect lines 16a' and 16b', as shown in Fig. 3, to open and close the signal path. “[A] device removal

event can be represented by deactivating the FET and opening this single<sup>5</sup> [sic] path. Likewise, a device insertion event can be represented by activating the FET and closing this signal path.” Ex. 1005, 10:29–32. Interrupting the passage of signals from the card detect lines 16a’ and 16b’ to the device controller 18 “trick[s] the device controller to take actions in response to the apparent removal of the device.” Ex. 1005, 9:43–51. Closing the switches “effectively ‘tricks’ the controller 18 into making a determination that the device 12 has been inserted into the socket 14.” *Id.* at 11:48–52.

## **X. UNPATENTABILITY ANALYSIS**

### **A. The Claims of the ’770 Patent**

98. I have been asked to review the claims of the ’770 Patent and to provide an opinion as to whether the subject matter of the claims would have been anticipated and/or obvious in light of the prior art.

99. The analysis below is presented on a claim-by-claim basis.

#### **Listing of Claims**

100. Below is a listing of the claims of the ’770 Patent that are being petitioned for *inter partes* review:

---

<sup>5</sup> One of ordinary skill in the art, reading the specification, would have recognized that “single path” contains a typographical error and should read “signal path.”



<b>Claim</b>	<b>Claim Language</b>	<b>Grounds of Invalidity</b>
1	<p>A system for reconfiguring a peripheral device having a first configuration connected by a computer bus to a host computer, the system comprising:</p> <p>a first circuit configured to download information for a second configuration from the host computer into the peripheral device over the computer bus; and</p> <p>a second circuit configured to electronically simulate a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device to said second configuration while supplying electrical power to said peripheral device.</p>	<p>1. APA in view of Yap 2. Michelson in view of PCCextend and Davis</p>
2	<p>The system of claim 1, wherein said first configuration is a generic configuration assigned to the peripheral device and said second configuration comprises any one of a plurality of unique manufacturer configurations.</p>	<p>1. APA in view of Yap and Michelson 2. Michelson in view of PCCextend and Davis</p>
3	<p>The system of claim 2, wherein said first circuit is configured to (i) read an identification code from the peripheral device and (ii) select said second configuration based on said identification code.</p>	<p>1. APA in view of Yap and Michelson 2. Michelson in view of PCCextend and Davis</p>
5	<p>The system of claim 1, wherein said computer bus comprises a Universal Serial Bus.</p>	<p>1. APA in view of Yap 2. Michelson in view of PCCextend and Davis and APA</p>
7	<p>The system of claim 5, wherein said information for said second configuration comprises (i) configuration data and (ii) an executable code.</p>	<p>1. APA in view of Yap 2. Michelson in view of PCCextend and Davis and APA</p>
10	<p>The system of claim 1, wherein said second circuit comprises a reset circuit configured to reset the first or second</p>	<p>1. APA in view of Yap 2. Michelson in view of PCCextend and Davis</p>

Claim	Claim Language	Grounds of Invalidity
	configuration of the peripheral device.	
11	<p>A method for reconfiguring a peripheral device having a first configuration connected by a computer bus to a host computer, the method comprising the steps of:</p> <p>(A) downloading information for a second configuration from the host computer into the peripheral device over the computer bus; and</p> <p>(B) electronically simulating a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device to said second configuration while supplying electrical power to said peripheral device.</p>	<ol style="list-style-type: none"> <li>1. APA in view of Yap</li> <li>2. Michelson in view of PCCextend and Davis</li> <li>3. USB 1.0 Specification</li> </ol>
12	12. The method of claim 11, wherein said first configuration comprises a generic configuration assigned to the peripheral device and said second configuration comprises any one of a plurality of unique manufacturer configurations.	<ol style="list-style-type: none"> <li>1. APA in view of Yap and Michelson</li> <li>2. Michelson in view of PCCextend and Davis</li> </ol>
13	13. The method of claim 11, wherein step (A) comprises: reading an identification code from the peripheral device, and selecting said second configuration based on said identification code.	<ol style="list-style-type: none"> <li>1. APA in view of Yap and Michelson</li> <li>2. Michelson in view of PCCextend and Davis</li> </ol>
15	15. The method of claim 11, wherein step (A) comprises communicating said information for the second configuration to the peripheral device over a Universal Serial Bus.	<ol style="list-style-type: none"> <li>1. APA in view of Yap</li> <li>2. Michelson in view of PCCextend and Davis and APA</li> </ol>
16	16. The method of claim 11, wherein said information for the second configuration comprises (i) configuration data and (ii) an executable code.	<ol style="list-style-type: none"> <li>1. APA in view of Yap</li> <li>2. Michelson in view of PCCextend and Davis</li> </ol>

Claim	Claim Language	Grounds of Invalidity
17	17. The method of claim 11, wherein step (B) comprises electronically resetting the configuration of the peripheral device, controllable by the peripheral device.	1. APA in view of Yap 2. Michelson in view of PCCextend and Davis
18	18. A system for reconfiguring a peripheral device having a configuration connected by a computer bus to a host computer, the system comprising:  a first circuit configured to detect the peripheral device connected to the computer bus; and  a second circuit configured to electronically simulate a physical disconnection and reconnection of the peripheral device to reset said configuration of said peripheral device while supplying electrical power to said peripheral device.	1. Yap 2. Michelson in view of PCCextend and Davis
19	19. The system of claim 18, wherein said computer bus comprises a Universal Serial Bus.	1. Yap 2. Michelson in view of PCCextend and Davis and APA
20	20. The system of claim 18, wherein said second circuit comprises a solid state switch.	1. Yap 2. Michelson in view of PCCextend and Davis

**B. Claim 11 is unpatentable under 35 U.S.C. § 102(b) as being anticipated by the USB Specification V1.0.**

101. The following analysis demonstrates, on a limitation-by-limitation basis, how claim 11 of the '770 patent is anticipated by the USB 1.0 Specification under 35 U.S.C. § 102(b). Claim 11 recites:

11. A method for reconfiguring a peripheral device having a first configuration connected by a computer bus to a host computer, the method comprising the steps of:

(A) downloading information for a second configuration from the host computer into the peripheral device over the computer bus; and

(B) electronically simulating a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device to said second configuration while supplying electrical power to said peripheral device.

102. For example, the USB specification explains that configuration and reset are part of the standard USB functionality for all devices.

USB devices present a standard USB interface in terms of their:

- Comprehension of the USB protocol
- Response to standard USB operations such as configuration and reset
- Standard capability descriptive information

Ex. 1013 at 29

103. Similarly, the USB specification explains that enumeration is a continuing function for USB systems. “Since the USB allows USB devices to attach to or detach from the USB at any time, bus enumeration for this bus is an on-going activity.” Ex. 1013 at 32.

104. Also, the USB specification explains that all USB systems include at least one USB hub and that a hub can attach and detach ports as well as control the power to ports and attached devices.

#### 4.1.1 Bus Topology

The Universal Serial Bus connects USB devices with the USB host. The USB physical interconnect is a tiered star topology. A hub is at the center of each star. Each wire segment is a point-to-point connection between the host and a hub or function, or a hub connected to another hub or function. Figure 4-1 illustrates the topology of the USB.

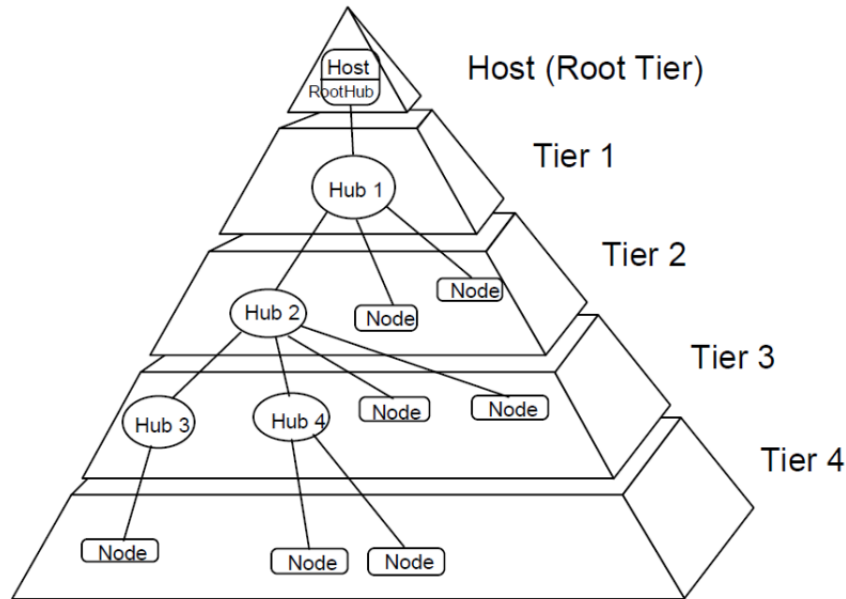


Figure 4-1. Bus Topology

Ex. 1013 at 28

Hubs are a key element in the plug-and-play architecture of USB. Figure 4-3 shows a typical hub. Hubs serve to simplify USB connectivity from the user's perspective and provide robustness at low cost and complexity.

Hubs are wiring concentrators and enable the multiple attachment characteristics of USB. Attachment points are referred to as ports. Each hub converts a single attachment point into multiple attachment points. The architecture supports concatenation of multiple hubs.

The upstream port of a hub connects the hub towards the host. Each of the other downstream ports of a hub allows connection to another hub or function. Hubs can detect attach and detach at each downstream port and enable the distribution of power to downstream devices. Each downstream port can be individually enabled and configured as either full or low speed. The hub isolates low speed ports from full speed signaling.

Ex. 1013 at 35

105. The USB specification discloses that each host port or hub port includes 15K $\Omega$  pull-down resistors on each of the D+ and D- data lines. Each connected full-speed device includes a 1.5K $\Omega$  pull-up resistor on the D+ data line. This is the same functionality disclosed in the '770 patent. This pull-up resistor circuit is used to detect the presence of a full-speed device on a port. This leads to

device enumeration and configuration. The device connects with a default configuration and then can be reconfigured according to USB 1.0 (Ex. 1013 at section at 34, section 4.8).

### 7.1.3 Signal Termination

The USB is terminated at the hub and function ends as shown in Figure 7-5 and Figure 7-6. Full speed and low speed devices are differentiated by the position of the pull-up resistor on the downstream end of the cable. Full speed devices are terminated as shown in Figure 7-5 with the pull-up on the D+ line. Low speed devices are terminated as shown in Figure 7-6 with the pull-up on the D- line.

The pull-up terminator is a  $1.5\text{ k}\Omega \pm 5\%$  resistor tied to a voltage source between 3.0 V and 3.6 V referenced to the local ground. The pulldown terminators are resistors of  $15\text{ k}\Omega \pm 5\%$  connected to their local ground.

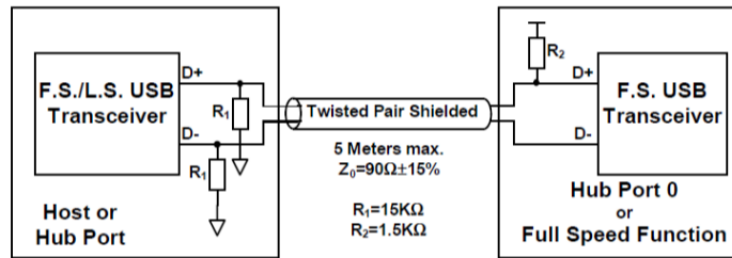


Figure 7-5. Full Speed Device Cable and Resistor Connections

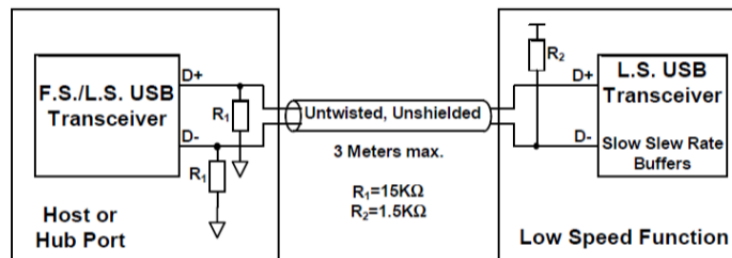


Figure 7-6. Low Speed Device Cable and Resistor Connections

Ex. 1013 at 114.

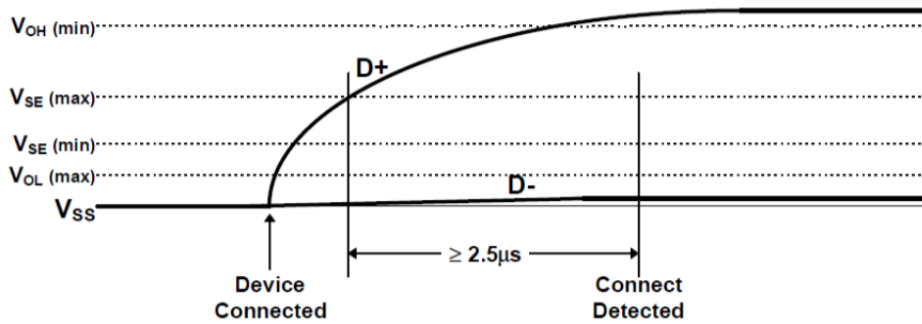


Figure 7-8. Full Speed Device Connect Detection

Ex. 1013 at 117.

106. This detection process is repeated whenever a device is powered down then powered on again, such as via a USB hub.

All ports on the downstream side of the host or a hub have pull-down resistors on both the D+ and D- lines. All devices have a pull-up resistor on one of the data lines on their upstream port. The type of device determines which data line has the pull-up resistor. Full speed devices have the pull-up on the D+ line (see Figure 7-5) and low speed devices have the pull-up on the D- line (see Figure 7-6). When a device is attached to hub or host but the data lines are not being driven, these resistors create a quiescent bias condition on the lines such that the data line with the pull-up is above 2.8 V and the other data line is near ground. This is called the idle state.

When no function is attached to the downstream port of the host or hub or the pull-up resistor on an attached device is not powered, the pull-down resistors will cause both D+ and D- to be pulled below the single-ended low threshold of the host or hub port. This creates a state called a single-ended zero (SE0) on the downstream port. A disconnect condition is indicated if an SE0 persists on a downstream port for more than 2.5  $\mu$ s (30 full speed bit times). Note that disconnect signaling applies only in an upstream direction (see Figure 7-7).

A connect condition will be detected when a device is connected to the host or hub's port, and one of the data lines is pulled above the single-ended high threshold level for more than 2.5  $\mu$ s (30 full speed data bit times). The data line that is high when the port state changes from disconnected to connected sets the idle state for this bus segment and determines whether the connected device is a full speed device or a low speed device. All signaling levels given in Table 7-1 are set for this network segment (and this segment alone) once the idle state is determined. Figure 7-8 shows a full speed device connection sequence and Figure 7-9 shows a low speed device connection sequence.

All hub ports start out in an implied disconnected state after power is applied to that port. If a device is connected to the port, the port goes through the connect sequence described above to detect the device type and set the port signaling characteristics (refer to Section 11.2.3).

Ex. 1013 at 117.

Hubs must differentiate between full speed and low speed devices when a device is connected to the bus or at power-up. Hubs detect whether a device is full or low speed when the hub port transitions from its disconnected to its disabled state. Devices attached to a hub are determined to be either full speed or low speed by detecting which data line (D- or D+) is pulled high. Low speed devices pull D- high, and full speed devices pull D+ high. Full speed signaling must not be transmitted to low speed devices. Doing so could cause low speed devices to mistakenly respond to full speed signaling and create a bus conflict. Communication between the host and the hub controller are always done using full speed signaling.

Ex. 1013 at 224.

Bus state evaluation is done at the end of the frame and is able to discriminate between the SE0, the differential 1 and 0 bus states. When no device is connected to a downstream hub port, its pull-down resistors pull both D+ and D- below  $V_{SE(min)}$ .

Connect/Disconnect detect can only be performed after  $V_{bus}$  is applied to the downstream port. (This requirement only affects hubs whose downstream ports are power switched.) When a device is connected, the bus state changes from the disconnected to the attach detect state. Low speed devices pull up D- to an SE1 and leave D+ at SE0. Full speed devices pull up D+ to an SE1 and leave D- at SE0. Each downstream hub port must be capable of detecting and differentiating between low speed and full speed device connections once a device is connected. The differential J and K states are undefined until a device is attached and the device's speed has been ascertained.

When a connect or disconnect occurs, it must be reflected in the hub status by the end of the frame in which the event occurred unless the hub is in the reset or suspend modes. A hub in the suspend mode is awakened by a connect or disconnect event and must be capable of reporting the event upon completion of resume. Upon coming out of reset, a hub must detect which downstream ports have devices connected to them. Connect and disconnect changes are reported on a per-port basis.

Ex. 1013 at 224.

107. After detection, the host reads the configuration information from the device. “When a USB device is attached, the following actions are undertaken: ...

7. The host reads the configuration information from the device by reading each configuration 0 to n.” Ex. 1013 at 169.

108. The same detection process is repeated after a reset.

<b>Default Address</b>	An address defined by the Universal Serial Bus Specification and used by a Universal Serial Bus device when it is first powered or reset. The default address is 00h.
------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

Ex. 1013 at 14.

All USB devices use the default address when initially powered or after the device has been reset. Each USB device is assigned a unique address by the host after attachment or after reset. A USB device maintains its assigned address while suspended.

Ex. 1013 at 168.

109. Claim 11 further requires “downloading information for a second configuration from the host computer into the peripheral device over the computer bus.” This step is disclosed in the USB 1.0 specification.

110. In general, a USB host must configure a USB device by downloading configuration information. In some cases, that configuration can persist after a



reset or disconnect while in some cases, a new configuration is performed after each reset or disconnect.

**Configuring Software** The host software responsible for configuring a Universal Serial Bus device. This may be a system configurator or software specific to the device.

Ex. 1013 at 14.

**Device Software** Software that is responsible for using a Universal Serial Bus device. This software may or may not also be responsible for configuring the device for use.

Ex. 1013 at 16.

Before the USB device's function may be used, the device must be configured. From the device's perspective, configuration involves writing a non-zero value to the device configuration register. Configuring a device or changing an alternate setting causes all of the status and configuration values associated with endpoints in the affected interfaces to be set to their default values. This includes setting the data toggle of any endpoint using data toggles to the value DATA0.

Ex. 1013 at 168.

When a USB device is attached to or removed from the USB, the host uses a process known as bus enumeration to identify and manage the device state changes necessary. When a USB device is attached, the following actions are undertaken:

1. The hub to which the USB device is now attached informs the host of the event via a reply on its status change pipe (refer to Chapter 11 for more information). At this point, the USB device is in the attached state and the port to which it is attached is disabled.
2. The host determines the exact nature of the change by querying the hub.
3. Now that the host knows the port to which the new device has been attached, the host issues a port enable and reset command to that port.
4. The hub maintains the reset signal to that port for 10 ms. When the reset signal is released, the port has been enabled and the hub provides 100 mA of bus power to the USB device. The USB device is now in the powered state. All of its registers and state have been reset and it answers to the default address.
5. Before the USB device receives a unique address, its default pipe is still accessible via the default address. The host reads the device descriptor to determine what actual maximum data payload size this USB device's default pipe can use.
6. The host assigns a unique address to the USB device, moving the device to the addressed state.
7. The host reads the configuration information from the device by reading each configuration zero to n. This process may take several frames to complete.
8. Based on the configuration information and how the USB device will be used, the host assigns a configuration value to the device. The device is now in the configured state and all of the endpoints in this configuration have taken on their described characteristics. The USB device may now draw the amount of Vbus power described in its configuration descriptor. From the device's point of view it is now ready for use.

When the USB device is removed, the hub again sends a notification to the host. Detaching a device disables the port to which it had been attached. Upon receiving the detach notification, the host will update its local topological information.

Ex. 1013 at 169.

- Self identifying peripherals, automatic mapping of function to driver, and configuration
- Dynamically attachable and reconfigurable peripherals

Ex. 1013 at 24.

<b>PERFORMANCE</b>	<b>APPLICATIONS</b>	<b>ATTRIBUTES</b>
<b>LOW SPEED</b> •Interactive Devices •10-100 Kb/s	Keyboard, Mouse Stylus Game peripherals Virtual Reality peripherals Monitor Configuration	Lower cost Hot plug-unplug Ease of use Multiple peripherals
<b>MEDIUM SPEED</b> •Phone, Audio, Compressed Video 500Kb/s - 10Mbps	ISDN PBX POTS Audio	Low cost Ease of use Guaranteed latency Guaranteed Bandwidth Dynamic Attach- Detach Multiple devices
<b>HIGH SPEED</b> •Video, Disk •25-500 Mb/s	Video Disk	High Bandwidth Guaranteed latency Ease of use

Figure 3-1. Application Space Taxonomy

Ex. 1013 at 24.

USB devices are divided into device classes such as hub, locator, or text device. The hub device class indicates a specially designated USB device which provides additional USB attachment points (refer to Chapter 11). USB devices are required to carry information for self-identification and generic configuration. They are also required at all times to display behavior consistent with defined USB device states.

Ex. 1013 at 34.

111. Configuration occurs following bus enumeration. “Since the USB allows USB devices to attach to or detach from the USB at any time, bus enumeration for this bus is an on-going activity.” Ex. 1013 at 32.

112. Configuration may be repeated with an alternate (second) configuration.

A USB device has one or more configuration descriptors. Each configuration has one or more interfaces and each interface has one or more endpoints. An endpoint is not shared among interfaces within a single configuration unless the endpoint is used by alternate settings of the same interface. Endpoints may be shared among interfaces that are part of different configurations without this restriction.

Once configured, devices may support limited adjustments to the configuration. If a particular interface has alternate settings, an alternate may be selected after configuration. Within an interface, an isochronous endpoint's maximum packet size may also be adjusted.

Ex. 1013 at 184.

Different operating system environments perform device configuration using different software components and different sequences of events. A specific operating system method is not assumed by the USB system. However, there are some basic requirements that must be fulfilled by any USB system implementation. In some operating systems, these requirements are met by existing host software. In others, the USB system provides the capabilities.

The USB system assumes a specialized client of the USB D, called a hub driver, which acts as a clearing house for addition of devices to and removal of devices from a particular hub. Once the hub driver receives such notifications, it will employ additional host software and other USB D clients, in an operating system specific manner, to recognize and configure the device. This model, shown in Figure 10-4 is the basis of the following discussion.

Ex. 1013 at 202.

Configuration Management services allow configuring software to replace a USB device configuration with another configuration from the set of configurations listed in the device. The operation succeeds if the data transfer rates given for all end points in the new configuration fit within the capabilities of the USB with the current schedule. If the new configuration is rejected, the previous configuration remains.

Ex. 1013 at 214.

113. Claim 11 further requires “electronically simulating a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device to said second configuration while supplying electrical power to said peripheral device.” This step is disclosed in the USB 1.0 specification.

114. The USB specification defines a reset process that electronically simulates a disconnection and reconnection. Devices are required to respond to the reset signal (an SE0 condition for  $> 2.5\mu\text{s}$ ) in the same manner as a physical disconnect and reconnect of a USB device and thereafter will adopt a second configuration based on the second set of configuration information as described with respect to the prior claim element.

Disconnect (Upstream only)	(n.a.)	D+ and D- < $V_{SE}(max)$ for $\geq 2.5 \mu s$
Connect (Upstream only)	(n.a.)	D+ or D- > $V_{SE}(max)$ for $\geq 2.5 \mu s$
Reset (Downstream only)	D+ and D- < $V_{SE}$ for $\geq 10 ms$	D+ and D- < $V_{SE}(min)$ for $\geq 2.5 \mu s$ (must be recognized within $5.5 \mu s$ ) <sup>3</sup>

Ex. 1013 at 115.

This section describes USB device states that are externally visible (see Figure 9-1). Note: USB devices perform a reset operation in response to a Reset request to the upstream port from the host. When reset signaling has completed, the USB device is reset.

Ex. 1013 at 165.

All ports on the downstream side of the host or a hub have pull-down resistors on both the D+ and D- lines. All devices have a pull-up resistor on one of the data lines on their upstream port. The type of device determines which data line has the pull-up resistor. Full speed devices have the pull-up on the D+ line (see Figure 7-5) and low speed devices have the pull-up on the D- line (see Figure 7-6). When a device is attached to hub or host but the data lines are not being driven, these resistors create a quiescent bias condition on the lines such that the data line with the pull-up is above 2.8 V and the other data line is near ground. This is called the idle state.

When no function is attached to the downstream port of the host or hub or the pull-up resistor on an attached device is not powered, the pull-down resistors will cause both D+ and D- to be pulled below the single-ended low threshold of the host or hub port. This creates a state called a single-ended zero (SE0) on the downstream port. A disconnect condition is indicated if an SE0 persists on a downstream port for more than 2.5  $\mu s$  (30 full speed bit times). Note that disconnect signaling applies only in an upstream direction (see Figure 7-7).

A connect condition will be detected when a device is connected to the host or hub's port, and one of the data lines is pulled above the single-ended high threshold level for more than 2.5  $\mu s$  (30 full speed data bit times). The data line that is high when the port state changes from disconnected to connected sets the idle state for this bus segment and determines whether the connected device is a full speed device or a low speed device. All signaling levels given in Table 7-1 are set for this network segment (and this segment alone) once the idle state is determined. Figure 7-8 shows a full speed device connection sequence and Figure 7-9 shows a low speed device connection sequence.

All hub ports start out in an implied disconnected state after power is applied to that port. If a device is connected to the port, the port goes through the connect sequence described above to detect the device type and set the port signaling characteristics (refer to Section 11.2.3).

Ex. 1013 at 116.

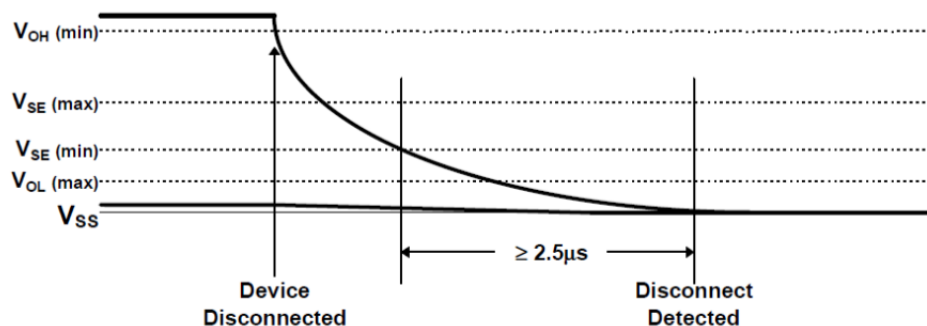


Figure 7-7. Disconnect Detection

Ex. 1013 at 117.

A reset is signaled downstream from a hub port on the bus by the presence of an extended SE0 at the upstream port of a device. After the reset is removed, the device will be in the attached, but not yet addressed or configured state (refer to Section 9.1). Note that reset signaling applies only in the downstream direction.

The reset signal can be generated by host command on any hub or host controller port. The reset signal must be generated for a minimum of 10 ms. The port that generated the reset will be sent to the logically disconnected state at the end of the reset. If a device is connected to the port, the bus pull-up resistor will determine the device type (low or full speed) and the port will end up in the disabled state (refer to Section 11.2.3).

An active device (powered and not in the suspend state) seeing a single-ended zero on its upstream port for more than 2.5  $\mu$ s may treat that signal as a reset, but must have interpreted the signaling as a reset within 5.5  $\mu$ s. A device that recognizes a reset from a SE0 between 32 and 64 full speed bit times or between 4 and 8 low speed bit times satisfies these requirements. The reset signal propagates through all enabled ports of any hubs downstream of the signaling port, but does not propagate through any ports that are disabled. A bus-powered hub that receives a reset on its root port removes power from all its downstream ports. After the reset is removed, all devices that received the reset are set to their default USB address and are in the unconfigured state. All ports on a hub that received a reset are disabled.

### Ex. 1013 at 119.

Bus state evaluation is done at the end of the frame and is able to discriminate between the SE0, the differential 1 and 0 bus states. When no device is connected to a downstream hub port, its pull-down resistors pull both D+ and D- below  $V_{SE(min)}$ .

Connect/Disconnect detect can only be performed after  $V_{bus}$  is applied to the downstream port. (This requirement only affects hubs whose downstream ports are power switched.) When a device is connected, the bus state changes from the disconnected to the attach detect state. Low speed devices pull up D- to an SE1 and leave D+ at SE0. Full speed devices pull up D+ to an SE1 and leave D- at SE0. Each downstream hub port must be capable of detecting and differentiating between low speed and full speed device connections once a device is connected. The differential J and K states are undefined until a device is attached and the device's speed has been ascertained.

When a connect or disconnect occurs, it must be reflected in the hub status by the end of the frame in which the event occurred unless the hub is in the reset or suspend modes. A hub in the suspend mode is awakened by a connect or disconnect event and must be capable of reporting the event upon completion of resume. Upon coming out of reset, a hub must detect which downstream ports have devices connected to them. Connect and disconnect changes are reported on a per-port basis.

### Ex. 1013 at 224.

USB devices must power on in such a manner that they do not drive D+ or D- (except with the pull-up resistor) during the reset process. This is required so the upstream hub can drive reset downstream and be assured that the downstream device will see the reset signaling.

### Ex. 1013 at 242.

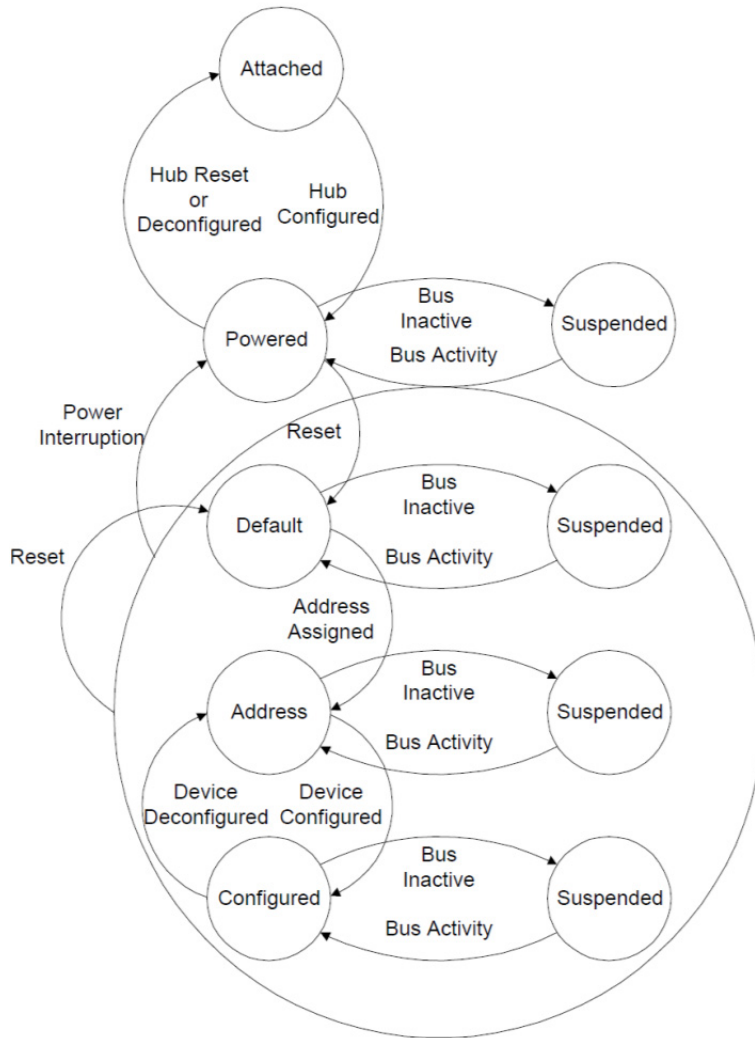


Figure 9-1. Device State Diagram

Ex. 1013 at 166.

Table 9-1. Visible Device States

Attached	Powered	Default	Address	Configured	Suspended	State
No	--	--	--	--	--	Device is not attached to USB. Other attributes are not significant.
Yes	No	--	--	--	--	Device is attached to USB, but is not powered. Other attributes are not significant.
Yes	Yes	No	--	--	--	Device is attached to USB and powered, but has not been reset.
Yes	Yes	Yes	No	--	--	Device is attached to USB and powered and has been reset, but has not been assigned a unique address. Device responds at the default address.
Yes	Yes	Yes	Yes	No	--	Device is attached to USB, powered, has been reset, and a unique device address has been assigned. Device is not configured.
Yes	Yes	Yes	Yes	Yes	No	Device is attached to USB, powered, has been reset, has unique address, is configured, and is not suspended. Host may now use the function provided by the device.
Yes	Yes	Yes	Yes	Yes	Yes	Device is, at minimum, attached to USB, has been reset, and is powered at the minimum suspend level. It may also have a unique address and be configured for use. However, since the device is suspended, the host may not use the device's function.

Ex. 1013 at 167.

When a USB device is attached to or removed from the USB, the host uses a process known as bus enumeration to identify and manage the device state changes necessary. When a USB device is attached, the following actions are undertaken:

1. The hub to which the USB device is now attached informs the host of the event via a reply on its status change pipe (refer to Chapter 11 for more information). At this point, the USB device is in the attached state and the port to which it is attached is disabled.
2. The host determines the exact nature of the change by querying the hub.
3. Now that the host knows the port to which the new device has been attached, the host issues a port enable and reset command to that port.
4. The hub maintains the reset signal to that port for 10 ms. When the reset signal is released, the port has been enabled and the hub provides 100 mA of bus power to the USB device. The USB device is now in the powered state. All of its registers and state have been reset and it answers to the default address.
5. Before the USB device receives a unique address, its default pipe is still accessible via the default address. The host reads the device descriptor to determine what actual maximum data payload size this USB device's default pipe can use.
6. The host assigns a unique address to the USB device, moving the device to the addressed state.
7. The host reads the configuration information from the device by reading each configuration zero to n. This process may take several frames to complete.
8. Based on the configuration information and how the USB device will be used, the host assigns a configuration value to the device. The device is now in the configured state and all of the endpoints in this configuration have taken on their described characteristics. The USB device may now draw the amount of Vbus power described in its configuration descriptor. From the device's point of view it is now ready for use.

When the USB device is removed, the hub again sends a notification to the host. Detaching a device disables the port to which it had been attached. Upon receiving the detach notification, the host will update its local topological information.

Ex. 1013 at 169.

115. The USB Specification also indicates that every USB system contains a hub. *See* Ex. 1013 at 28, 35 as cited above.

116. USB hubs are disclosed as being able to disconnect and reconnect device power using software. This uses a switch to simulate a disconnect and reconnect of the device.



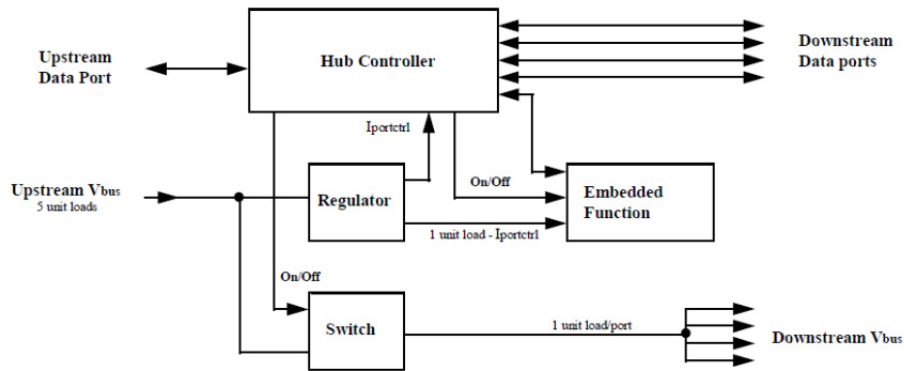


Figure 7-23. Compound Bus-powered Hub

Power to downstream ports must be switched. The hub controller supplies a software controlled on/off signal from the host, which is in the “off” state when the device is powered up or after reset signaling. When switched to the “on” state, the switch implements a soft turn-on function which prevents excessive transient current from being drawn from the upstream port. The voltage drop across the upstream cable, connectors, and switch in a bus-powered hub must not exceed 350 mV at maximum rated current.

Ex. 1013 at 132.

117. Since each and every element of claim 11 is disclosed in the USB specification, claim 11 is anticipated.

**C. Claims 1, 5, 7, 10, 11, and 15–17 are unpatentable under 35 U.S.C. § 103(a) as being obvious over the APA in view of Yap**

**1. Independent Claim 1**

118. Based on my review and analysis, it is my opinion that the combination of the APA and Yap discloses the subject matter of independent claim 1 and renders it obvious.

119. Claim 1 recites “A system for reconfiguring a peripheral device having a first configuration connected by a computer bus to a host computer.” As discussed above in Sections VI and IX, the APA discloses connecting a peripheral device to a host computer through a standard USB computer bus. Ex. 1001, 1:50–2:19; Fig. 1; 4:15–33.

120. The APA further discloses that the peripheral device in the prior art can have a first configuration. *Id.* at 1:66–2:19. It specifically states that “During the [enumeration process] query, a data table stored in the peripheral device, which contains the particular peripheral device’s configuration information, is read from the peripheral device into the host computer’s memory.” *Id.* at 2:10–14. In my opinion, the disclosure of a data table containing the peripheral device’s configuration information is an example of a first configuration. That is, a first set of configuration information would define a first configuration for the peripheral device. Thus, the APA teaches a peripheral device having a first configuration connected by a computer bus to a host computer.

121. The APA further discloses that it was known in the USB prior art for the host “to alter the configuration or personality of a peripheral device, such as downloading new code or configuration information into the memory of the peripheral device.” Ex. 1001, 2:24–29. Altering the configuration or personality of the device is an example of reconfiguring a peripheral device. Thus, the APA discloses reconfiguring of the peripheral device having the first configuration connected by the computer bus to the host computer.

122. Claim 1 further recites “a first circuit configured to download information for a second configuration from the host computer into the peripheral device over the computer bus.”

123. The APA teaches that it was known to download new code or configuration information into the peripheral device over the computer bus, but a problem with known systems and methods was that this required the host computer system to detect a peripheral device connection or a disconnection and then a reconnection:

In a serial bus system, such as the USB, the only opportunity for associating software device drivers with a peripheral device is at the time when the peripheral device is plugged into the USB and the enumeration process occurs. Thus, **to alter the configuration or personality of a peripheral device, such as downloading new code or configuration information into the memory of the peripheral device, the host computer system must detect a peripheral device connection or a disconnection and then a reconnection.**

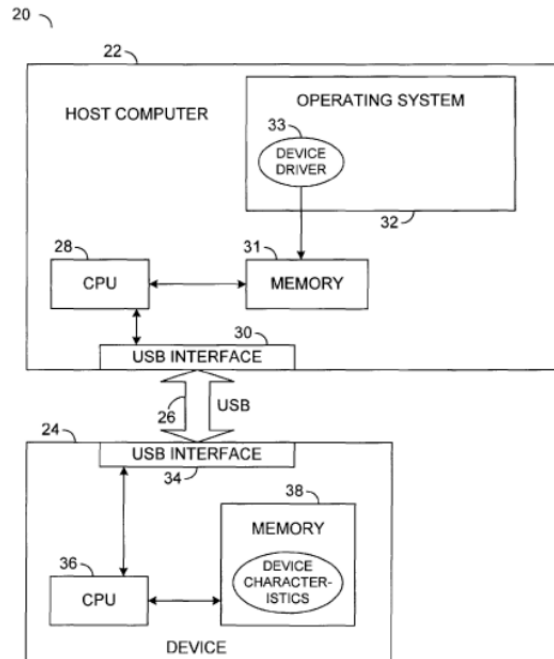
...

Thus, there is a need for a system and method for interfacing to a universal serial bus which avoids these and other **problems of known systems and methods**, and it is to this end that the present invention is directed.

Ex. 1001, 2:20–40 (emphasis added).

124. In my opinion, such downloading requires a “circuit configured to download information.” The disclosure of “new code or configuration information” in the APA is an example of “information for a second configuration” which is downloaded from the host computer into the peripheral device over the

computer bus. The new code or configuration information can alter the personality of the peripheral device to change its requirements or capabilities. The peripheral device with the altered requirements and capabilities has a second configuration. Further, one of ordinary skill in the art would have understood from the Background that the downloading can take place over the computer bus because it states that a host computer must detect a “peripheral device connection or a **disconnection and then a reconnection**” to alter the device’s personality by downloading new code or configuration information. This language suggests that the peripheral device was connected to the host for downloading and subsequently disconnected and reconnected. Further, because the prior art Fig. 1 (reproduced below) shows only a single computer bus 26 connected to the device, it would have been obvious to one of ordinary skill in the art for the downloading in the Background to occur over the computer bus. Thus, the APA discloses the first circuit as recited in claim 1.

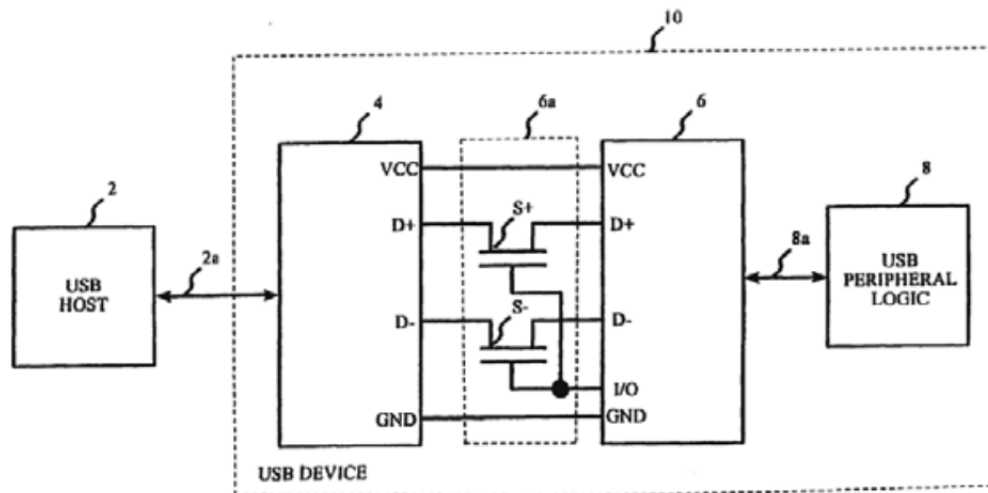


**FIG. 1**  
(PRIOR ART)

125. Claim 1 further recites “a second circuit configured to electronically simulate a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device to said second configuration.”

126. The APA discloses that it was known that altering the configuration of a peripheral device connected to a host computer required the host to detect a disconnection and then a reconnection to cause the host computer to recognize the new configuration. Ex. 1001, 2:20–29. The APA also states that this was one of the “problems of known systems and methods.” Ex. 1001, 2:37–40. It was a problem because it required a physical disconnection and reconnection of the peripheral device. Ex. 1001, 2:13–17.

127. Yap discloses a solution to the above problem. Yap teaches a circuit that is configured to electronically simulate the physical disconnection and reconnection of a peripheral device connected by a USB to a host computer to reconfigure the peripheral device. Ex. 1002, Fig. 2 (reproduced below) and Fig. 3.



*Fig. 2*

128. Yap describes that a malfunction may occur in a USB device, wherein after the device is configured, the host may terminate the function of the USB device for not communicating with the host computer a number of times (called a “brown out” condition) and not try to re-establish communications. Ex. 1002, 1:43–58.

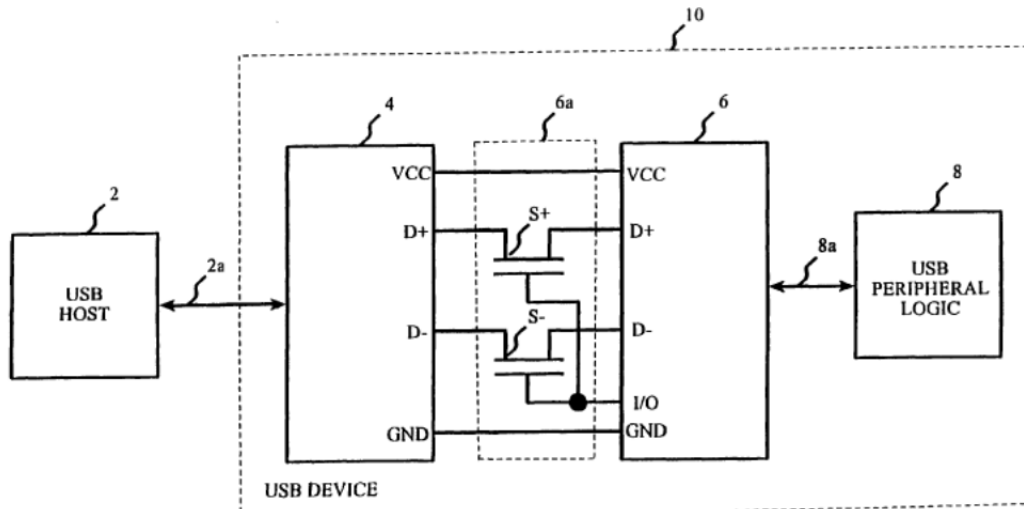
“When this occurs, (1) the user may have to re-boot the USB device or *physically disconnect and then re-connect the USB device to allow the host computer to recognize and then re-configure the USB device . . .*” Ex. 1002, 1:58–2:3

(emphasis added). Yap appreciates that this “method defeats the whole purpose of

plug-and-play technology,” where devices are automatically configured by the host computer. *Id.* at 1:66–67.

129. Yap relates to recovering from such a malfunction “without a need to re-boot the USB device or physically disconnect and then re-connect the USB device.” Ex. 1002, 2:20–24. Thus, Yap expressly discloses that the disadvantage of having to physically disconnect and reconnect the USB device to allow the host computer to recognize and reconfigure the USB device is that it may be inconvenient.

130. Yap teaches a circuit that is configured to electronically simulate the physical disconnection and reconnection of the peripheral device to allow the USB host to reconfigure the USB device using switching devices S+ and S- in the data lines D+ and D-, respectively, as shown in Figure 2 (reproduced below) and 3:60–4:23:



*Fig. 2*

In FIG. 2, a first embodiment of the USB device 10 of FIG. 1 further includes switching devices S+ and S-, such as transistors, contact switches, etc., coupled to positive data (D+) and negative data (D-) lines of the signal lines 6a. . . . Accordingly, when the USB micro-controller 6 drives the I/O pin to an appropriate logic state, the D+ and D- data lines may be opened or shorted via switching devices S+ and S-. **By disconnecting the D+ and D- data lines via switching devices S+ and S-, a physical removal of the USB device 10 may be simulated in order to allow the USB host to re-configure the USB device 10 during a brown out condition. . . .** Firmware in the USB micro-controller 6 keeps the data lines connected via switching devices S+ and S- during normal operation. However, **when a brown out condition is detected, as will be described later, the USB micro-controller 6 opens the data lines via the switching devices S+ and S- for a duration greater than 2.5 micro-seconds and then reconnects them again. This**



**procedure, for example, emulates the disconnect and re-connect procedure as specified in the USB specification v1.0, page 116.**

*Id.* at 3:60–4:23 (Emphasis added.); *see also* Fig. 3 (showing embodiment with switching devices S+ and S- of Figure 2 within the USB microcontroller); Ex. 1002, 4:24–38.

131. Thus, Yap discloses a circuit configured to electronically simulate the disconnect and reconnect procedure of the USB Specification 1.0 as a solution to the problem of requiring a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device. Ex. 1002, 2:20–25; Ex. 1013 at 116–169. Like the circuit shown and described in the '770 Patent, the simulation circuit in Yap changes the state of bus data lines D+ and D- and is thus an example of a circuit that simulates a disconnection and reconnection over the computer bus. Yap discloses that a USB disconnect and reconnect would result in a reconfiguration of the peripheral device. Ex. 1002, 3:60–4:23; *see also* Ex. 1001, 1:66–2:4 and 2:20–29; Ex. 1013 at 116, 169.

132. In my opinion, one of ordinary skill in the art would have been motivated to modify the system described in the APA to include the electronic disconnect and reconnect circuit of Yap because Yap discloses that such circuit solves the same problem described by the APA - namely, the inconvenience of having to physically disconnect and reconnect a peripheral device to reconfigure the device. *See* Ex. 1002, 1:66–2:3 (“[T]he first method defeats the whole purpose of

plug-and-play technology”). Further, as Yap indicates, using the switches to emulate a disconnection and reconnection furthers the aim of plug and play technology. Ex. 1002, 1:67. Additionally, such modification uses a known technique (e.g., placing a switch on a data line) to achieve a predictable result (e.g., simulation of a disconnect and reconnect over a bus). In my opinion, applying the switches in Yap to the teachings of the APA would have resulted in a system that includes a circuit configured to electronically simulate a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device to a second configuration. Modifying the system of the APA to include a circuit that electronically simulates a disconnection and reconnection of the peripheral device as taught by Yap would cause the peripheral device to be reconfigured to a second configuration (i.e., an altered configuration) based on a second set of configuration information (i.e., the new code and configuration information downloaded to the device) because the APA states that this is what happens when the host computer detects a peripheral device disconnection and then a reconnection. Ex. 1001, 2:24–29.

133. Claim 1 further recites that the second circuit is configured to perform the electronic simulation “while supplying electrical power to said peripheral device.”

134. Yap teaches that the electronic simulation of a physical disconnection and reconnection is performed while electrical power is supplied to the peripheral device. More specifically, Figs. 2 and 3 in Yap (Ex. 1002) show that the USB micro-controller 6 and USB peripheral logic 8 remain powered by VCC from the USB host 2 during the electronically simulated physical disconnection and reconnection, because no switch is placed on the VCC line. Further, because the micro-controller in the peripheral device is controlling the switches S+ and S- during the simulated disconnection and re-connection (Ex. 1002, 4:3–9), the peripheral device must remain powered. For instance, the micro-controller would need power to turn on the switches S+, S- to simulate a device reconnect. A person of ordinary skill in the art would understand the teaching from Yap figures 2 and 3 that show power and ground coming from the USB connector to teach such a bus-powered device.

135. Thus, it is my opinion that claim 1 would have been obvious over the combination of APA and Yap.

## **2. Dependent Claim 5**

136. Claim 5 depends on claim 1, which is addressed above in Section X.C.1, and further recites “wherein said computer bus comprises a Universal Serial Bus.”

137. The APA discloses this feature. For example, the APA discloses that it was known in the prior art that “a peripheral device is first connected to the USB and the host computer through a standard USB communications port . . . .” Ex. 1001, 1:66–2:4; Fig. 1. The APA also discloses that it was known in the prior art to download new configuration information to the peripheral device over a USB. Ex. 1001, 2:24–29. That is, a USB interface is the only interface shown to exist between the host computer and the peripheral device in the prior art system shown in Fig. 1 of the ’770 patent, and the Background states that the host must detect a “disconnection and reconnection” of the device to alter the configuration by downloading, one of ordinary skill in the art would understand that the prior art downloading by the host referred to in the Background must be able to take place over the USB interface (*i.e.*, a Universal Serial Bus (USB) and port), and hence the prior art downloading to the peripheral device must also be able to take place over the USB interface. Yap also discloses an electronic circuit for electronically simulating a disconnect and reconnect over a Universal Serial Bus and port. Ex. 1002, Figs. 2–3. Thus, the combination of the APA and Yap would include a Universal Serial Bus and port as claimed. As discussed above with respect to independent claim 1 in Section X.C.1, it would also have been obvious to one of ordinary skill in the art for the downloading in the Background to occur over the USB and port. Further, it would have been obvious to communicate the

information to the peripheral device over the universal serial bus and port in the APA because, according to the APA, USB is a type of communications port over which data may be communicated. *See, e.g.*, Ex. 1001, 1:50–65; 4:23-26, 51–56; Fig. 1.

138. Thus, it is my opinion that claim 5 would have been obvious over the APA and Yap.

### 3. Dependent Claim 7

139. Claim 7 depends on claim 5, which is addressed above in Section X.C.2, and further recites “wherein said information for said second configuration comprises (i) configuration data and (ii) an executable code.”

140. The APA discloses configuration data and executable code. For example, the APA discloses that the following was a known problem in the prior art:

Thus, to alter the configuration or personality of a peripheral device, such as downloading **new code or configuration information** into the memory of the peripheral device, the host computer system must detect a peripheral device connection or a disconnection and then a reconnection.

Ex. 1001, 2:24–29 (emphasis added), 2:36–40.

141. The “new code” is an example of “executable code” and the new “configuration information” is an example of “configuration data.” Thus, the combination of APA and Yap would include a circuit configured to download configuration data and/or executable code. It would have been obvious to one of

ordinary skill in the art that one or both of these types of configuration information could have been downloaded, depending on what type of configuration changes were desired. *See, e.g.*, Snyder (Ex. 1015), at 2:41–44; 3:64–4:11; 5:11–22; 6:24–30; 7:51–67. *See also* Bedingfield (Ex. 1020), at 2:6–66; 5:42–55; 4:27–3:33; 7:28–33. Thus, claim 7 would have been obvious over the combination of APA and Yap.

#### 4. Dependent Claim 10

142. Claim 10 depends on claim 1, which is addressed above in Section X.C.1, and further recites “wherein said second circuit comprises a reset circuit configured to reset the first or second configuration of the peripheral device.”

143. Yap discloses this feature. For example, Yap discloses:

By disconnecting the D+ and D- data lines via switching devices S+ and S-, a **physical removal of the USB device 10 may be simulated in order to allow the USB host to re-configure the USB device 10** during a brown out condition. . . . Firmware in the USB micro-controller 6 keeps the data lines connected via switching devices S+ and S- during normal operation. However, when a brown out condition is detected, as will be described later, **the USB micro-controller 6 opens the data lines via the switching devices S+ and S- for a duration greater than 2.5 microseconds and then reconnects them again. This procedure, for example, emulates the disconnect and re-connect procedure as specified in the USB**

**specification v1.0, page 116.<sup>6]</sup>**

Ex. 1002, 4:6–23 (Emphasis added).

144. The switch circuit that emulates the disconnect and re-connect procedure in Yap is a reset circuit configured to reset the first or second configuration of the peripheral device because it is essentially the same structure as disclosed in the '770 Patent. *See* Ex. 1001, Fig. 4; Ex. 1002, Figs. 2 and 3. Also, Yap discloses “a physical removal of the USB device 10 may be simulated in order to allow the USB host to re-configure the USB device 10.” Ex. 1002, 4:6–10. Yap further discloses “[T]he USB micro-controller 6 opens the data lines via the switching devices S+ and S- for a duration greater than 2.5 micro-seconds and then reconnects them again. This procedure, for example, emulates the disconnect and re-connect procedure as specified in the USB specification v1.0, page 116.” *Id.* at 4:16–23. The USB v 1.0 Specification is incorporated by reference in Yap. Ex. 1002, 1:38–41. (*See* the analysis of the USB specification above.) Section 9.1.2 of the USB v1 .0 Specification, for instance, teaches that attachment of a USB device causes a process known as a bus enumeration, during which the host issues a reset command to reset the configuration of the device. Ex. 1013, p. 169. Thus, a circuit which electronically simulates a physical disconnection and reconnection, as

---

<sup>6</sup> As cited previously, page 116 of the USB specification (Ex. 1013) discloses a reset procedure.

taught in Yap, includes a reset circuit configured to reset the configuration of the peripheral device. Also, Yap relates to a USB system, and it was well-known at the time that opening a data line for a duration greater than 2.5 micro-seconds in a USB system causes a reset of the current configuration (e.g., a first or second configuration). Ex. 1013, pp. 116–119, 223, 240.

145. As such, the proposed combination of the APA and Yap discloses a reset circuit configured to reset the first or second configuration of the peripheral device.

146. Thus, it is my opinion that claim 10 would have been obvious over the APA and Yap.

## **5. Independent Claim 11**

147. Claim 11 is a method claim version of system claim 1. That is, claim 11 recites method steps performed by the first and second circuits in claim 1 and recited in that claim. Based on my review and analysis, it is my opinion that the combination of the APA in the '770 patent and Yap renders the subject matter of independent claim 11 obvious for the same reasons set forth above in Sections X.C.1 with respect to claim 1.

148. Thus, it is my opinion that claim 11 would have been obvious over the APA and Yap.



## 6. Dependent Claim 15

149. Claim 15 depends on claim 11, which is addressed above in Section X.C.5, and further recites “wherein step (A) comprises communicating said information for the second configuration to the peripheral device using a Universal Serial Bus.” The APA teaches or suggests this feature. First, as acknowledged in the APA, it was known for a host computer to communicate with the peripheral device using the USB. Ex. 1001, 4:23–26; 5:1–56. Second, one of ordinary skill in the art would have understood the terms “communicating” and “downloading” to be equivalent in view of the specification and claims. *See, e.g.*, Ex. 1001, 4:14–34, 5:1–62; claims 6, 15. As noted above with respect to claim 5 in Section X.C.2, the APA discloses downloading (i.e., communicating) information for a second configuration to a peripheral device over a Universal Serial Bus. *See supra* Section X.C.2. As such, the combination of the APA and Yap would include this feature. Further, it would have been obvious to communicate the information to the peripheral device over the universal serial bus and port in the APA because, according to the APA, USB is a type of communications port over which data may be communicated. *See, e.g.*, Ex. 1001, 1:50–65; 4:15–28, 51–56; Fig. 1. Therefore, claim 15 would have been obvious over the APA and Yap.

## **7. Dependent Claim 16**

150. Claim 16 depends on claim 11, which is addressed above in Section X.C.5, and further recites “wherein said information for the second configuration comprises (i) configuration data and (ii) an executable code.” Thus, claim 16 is a method version of system claim 7. As discussed above with respect to claim 7 in Section X.C.3, the APA discloses downloading configuration data and/or executable code, and it would have been obvious to download one or both in the combination of the APA and Yap. Thus, claim 16 would have been obvious over the APA and Yap for the same reasons discussed with respect to claim 7 in Section X.C.3.

## **8. Dependent Claim 17**

151. Claim 17 depends on claim 11, which is addressed above in Section X.C.5, and further recites “wherein step (B) comprises electronically resetting the configuration of the peripheral device, controllable by the peripheral device.” Claim 17 is a method claim equivalent to system claim 10, and additionally calls for resetting to be “controllable by the peripheral device.” As discussed above with respect to claim 10 in Section X.C.4, Yap teaches or suggests electronically resetting the configuration of the peripheral device.

152. Claim 17 further recites that the resetting is “controllable by the peripheral device.” However, this feature is also shown by Yap. For example, in the electronic disconnect and reconnect circuit disclosed in Yap, “the USB

micro-controller 6 opens the data lines via the switching devices S+ and S-....” Ex. 1002, 4:18–23. In addition, Figs. 2 and 3 show the control lines for the switching devices S+ and S- connected to or disposed within the USB micro-controller 6, which is part of the peripheral device 10. Ex. 1002, Figs 2 and 3. As such, the electronic disconnect and reconnect, and concomitantly the “reset” discussed above in connection with claim 10, is “controllable by the peripheral device.”

153. Thus, claim 17 would have been obvious over the APA and Yap.

**D. Claims 2, 3, 12, and 13 are unpatentable under 35 U.S.C. § 103(a) as being obvious over APA in view of Yap and Michelson**

**1. Dependent Claim 2**

154. Dependent claim 2 depends from independent claim 1, which is addressed above in Section X.C.1, and recites that “said first configuration is a generic configuration assigned to the peripheral device and said second configuration comprises any one of a plurality of unique manufacturer configurations.” Michelson discloses this feature.

155. Michelson discloses a PCMCIA card that may be reprogrammed with data stored on a host computer through a standard PCMCIA bus. Ex. 1003 at Abstract. The PCMCIA card includes a Card Information Structure (CIS) EEPROM 30 that stores CIS data which is read and used by the host computer to configure the computer and the card to operate together and to load application software which causes FPGA programming data to be downloaded to the card. *Id.*

at 2:17–44 and 3:34–4:8. The configuration defined at least in part by the CIS data is an example of a generic configuration in the sense that it provides general configuration information required to configure the host computer and card to operate together. The CIS data may include identification of the card manufacturer and card identification (ID) number. Ex. 1003, 3:49–51.

156. Michelson distinguishes the CIS data from FPGA programming data, which is downloaded in response to the CIS data, and which is “design-specific data” that controls the operation of the functional hardware. Ex. 1003, 1:50–57. A device that only contains the CIS data has a first configuration which comprises a generic configuration because it is not yet functionally programmed. Michelson discloses that the CIS data, including the manufacturer identification, is used by the host computer to select an FPGA programming data file that corresponds to “a particular application for the PCMCIA card.” Ex. 1003, 3:63–64. Since the downloaded FPGA programming data is selected based in part on the manufacturer identification, it would have been obvious for the downloaded data to comprise a unique manufacturer configuration. Michelson also discloses that “[t]he FPGA programming data files can be supplied with the PCMCIA card....” (Ex. 1003, 6:61–62), *i.e.*, by the manufacturer, which one of ordinary skill in the art would understand to mean that the programming data file may constitute a unique manufacturer configuration. A plurality of unique manufacturer configurations

could exist because it would have been obvious for different manufacturers to each provide its own unique FPGA programming data. Thus, once the device in Michelson is programmed with a downloaded application from the manufacturer, it has one of a plurality of unique manufacturer configurations.

157. It would have been obvious to modify the APA and Yap to include an initial generic configuration and a downloadable unique manufacturer configuration as taught by Michelson in order to allow a user to access a large number of programming data files to program and reprogram the peripheral device without the need for storing the files on the device. *See, e.g.*, Ex. 1003, 1:65–67 (“The EPROM(s) required to store the FPGA programming data generally consumes a large amount of the PCMCIA card real estate”). This involves a simple substitution of known features in a known manner to achieve predictable results.

158. Further, it would have been obvious to modify the APA and Yap so that the first configuration is a generic configuration and the second configuration is a unique manufacturer configuration, as taught by Michelson, because all three references relate to configuring peripheral devices connected to a host computer by a computer bus and port, and the modification involves a simple substitution of known features (*i.e.*, generic and unique manufacturer sets of configuration information) in a known manner to achieve predictable results. It would also have been obvious for a first configuration to include a generic configuration so that microcontroller

manufacturers could sell microcontrollers with a generic configuration for use in a variety of peripheral devices to provide only basic functionality, such as allowing the device to communicate with a host computer, and manufacturers or users of the peripheral devices could thereafter update the configuration to include a unique manufacturer configuration providing specific functionality. In fact, it was well known to do so at the time of the invention. *See, e.g.*, Michelson (Ex. 1003), 1:50–2:45; Balbinot (Ex. 1014), 3:12–23; 2:21–59; Quinnell (Ex. 1016), p. 48 (describing “enumeration code that Intel provided [that] allows the evaluation board to respond to the USB host’s setup commands and receive an address assignment” and “[t]o turn the evaluation board into a peripheral device, I would have to program the 82930 [an Intel USB microcontroller].”).

159. Thus, it is my opinion that claim 2 would have been obvious over the combination of the APA, Yap, and Michelson.

## **2. Dependent Claim 3**

160. Dependent claim 3 depends on claim 2, which is addressed above in Section X.D.1. Claim 3 further recites “wherein said first circuit is configured to (i) read an identification code from the peripheral device and (ii) select said second configuration based on said identification code.” Michelson discloses this feature.

161. For example, Michelson discloses that the processor 22 of the host computer reads CIS data which identifies the PCMCIA card to enable the processor

to select the appropriate FPGA programming data (configuration software) “that corresponds to a particular application for PCMCIA card 14.” Ex. 1003, 3:61–66. The CIS data includes a card manufacturer identification and card identification (ID) number. Ex. 1003, 3:49–54. The card manufacturer and/or the card (ID) number read from the CIS are examples of an identification code under a broadest reasonable interpretation of the term, and the downloaded programming data is selected based on the identification information. It would have been obvious to modify the combination of the APA and Yap to read an identification code from the peripheral device and select the second configuration based on the identification code as disclosed by Michelson to ensure that an appropriate configuration is selected. Moreover, the USB specification itself as incorporated into Yap, discloses reading an identification code from the peripheral device and selecting a second configuration based on that identification code. *See e.g.* ex. 1013 at 24, 169, 184, and 214

162. Thus, the claim 3 would have been obvious over the APA, Yap, and Michelson.

### **3. Dependent Claim 12**

163. Claim 12 depends from independent claim 11, which is addressed above in Section X.C.5. Claim 12 further recites “wherein said first configuration comprises a generic configuration assigned to the peripheral device and said second

configuration comprises any one of a plurality of unique manufacturer configurations.” Thus, claim 12 is a method version of system claim 2. As discussed above with respect to claim 2 in Section X.D.1, this feature is disclosed by Michelson. *E.g.*, Ex. 1003, 3:34–4:8.

164. Thus, claim 12 would have been obvious over APA, Yap, and Michelson for the same reasons disclosed above in Section X.D.1.

#### **4. Dependent Claim 13**

165. Claim 13 depends from claim 12, which is addressed above in Section X.D.3. Claim 13 further recites “wherein step (A) comprises: reading an identification code from the peripheral device, and selecting said second configuration based on said identification code.” Thus, claim 13 is a method version of system claim 3. As discussed above with respect to claim 3 in Section X.D.2, this feature is disclosed and/or suggested by Michelson and the USB specification incorporated into Yap.

166. Thus, claim 13 would have been obvious over the APA, Yap, and Michelson for the same reasons discussed above with respect to claim 3 in Section X.D.2.



**E. Claims 1–3, 10, 11–13, 16–18, and 20 are unpatentable under 35 U.S.C. § 103(a) as being obvious over Michelson in view of PCCextend and Davis**

**1. Independent Claim 1**

167. As noted above, claim 1 recites a system for reconfiguring a peripheral device having a first configuration connected by a computer bus to a host computer.

168. Michelson “relates to programming and reprogramming the hardware configuration of a (PCMCIA) card.” Ex. 1003 1:7–16. “PCMCIA cards are typically used to add functionality or memory to a personal, portable, or desktop computer (i.e., host computer)” (1:13–15) and so are “peripheral devices.” Thus, Michelson discloses a system for reconfiguring a peripheral device. *Id.*

169. Michelson states that a “typical PCMCIA card includes a standard PCMCIA connector connected to a PCMCIA interface circuit through a standard PCMCIA bus.” Ex. 1003, 1:28–30. “[T]he host computer includes a PCMCIA adapter circuit coupled to a PCMCIA host socket which is mechanically and electrically connected to a PCMCIA card connector on the PCMCIA card” (*Id.* at 2:17–20), and “PCMCIA card 14 card connector 28 is inserted in PCMCIA host socket 18 of host computer 12” (*Id.* at 3:34–37). “Through a standard PCMCIA bus (*i.e.*, PCMCIA address lines 62, data lines 64, and control lines 66) connected to PCMCIA connector 28”, the peripheral PCMCIA device receives data from the host computer. *Id.* at 4:13–23. The PCMCIA bus is an example of a computer bus. A

“port” is an interface between a host computer and a peripheral, and the socket is an example of a port of the host computer. Thus, Michelson teaches a PCMCIA peripheral device connected by a PCMCIA bus to a port of a host computer.

170. Further, the PCMCIA card in Michelson has a first configuration as explained above in regard to claim 2 in Section X.D.1. Thus, Michelson discloses a system for reconfiguring a peripheral device having a first configuration connected by a computer bus to a host computer.

171. Claim 1 further recites “a first circuit configured to download information for a second configuration from the host computer into the peripheral device over the computer bus.”

172. Michelson teaches this feature. After the host computer 12 and PCMCIA card 14 are configured by information in the CIS, as discussed previously, programming data is downloaded by a processor (which is an example of a first circuit) to the PCMCIA card:

Processor 22 then executes (step 50) the application software 40 resident in host memory 24 that corresponds to PCMCIA card 14. The application software 40 causes the processor to either select a default FPGA programming data file 42 from host memory 24 that corresponds to a particular application for PCMCIA card 14 or request input from the user as to which FPGA programming data file 42 is to be selected from host memory 24. **Processor 22 then sends (step 52) the data from the selected FPGA programming data**

**file 42 through PCMCIA adapter 16 to PCMCIA interface chip 26. Interface chip 26 then programs (step 54) a field programmable gate array (FPGA, not shown in FIG. 1) within card controller 32 by loading the data from the FPGA programming data file 42 into the FPGA.** Where the application software causes the processor to select a default data programming file, PCMCIA card 14 and host computer 12 are made operable (step 56) without user intervention.

Ex. 1003, 3:59–4:8 (emphasis added); steps 50–56 of Figure 2.

173. The information is downloaded over the bus: “Through a standard PCMCIA bus (i.e., PCMCIA address lines 62, data lines 64, and control lines 66) connected to PCMCIA connector 28, interface chip 26 receives FPGA programming data from host computer 12 (FIG. 1).” *Id.* at 4:13–17. The default or user-selected FPGA programming data file 42, which is downloaded from the host computer into the FPGA in the PCMCIA card over the PCMCIA bus in Michelson is an example of “information for a second configuration.” The discussion above with respect to FPGA programming data is sufficient to show reconfiguring to a second configuration, but it was also well known at the time that a host could change the CIS in a PCMCIA peripheral device, which would also result in a reconfiguration to a second configuration. Ex. 1020, 7:28–33 (“In addition, in the prior art, the PCMCIA attribute structure is typically pre-defined and non-changeable, e.g., a read-only memory (ROM) is used to provide the PCMCIA attribute structure.

However, the use of shared memory 130 allows for a software definable PCMCIA card information structure that can be dynamically altered by CPU 170.”). Thus, Michelson discloses a first circuit configured to download information for a second configuration from the host computer into the peripheral device over the computer bus.

174. Claim 1 further recites “a second circuit configured to electronically simulate a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device to said second configuration”

175. Michelson discloses that, after the processor downloads FPGA programming data file 42 through PCMCIA adapter 16 to PCMCIA interface chip 26, the FPGA is “reset” to enable reprogramming:

Interface chip 26 initiates FPGA 60 programming through FPGA programming circuit 68, which drives reset line 63 and reprogram line 65, and completes FPGA 60 programming by loading the FPGA programming data into FPGA 60 through peripheral data lines 72.

*Id.* at 4:17–22 (emphasis added).

176. Thus, Michelson teaches the limitation of “a second circuit configured to electronically simulate a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device to said second configuration” in at least two ways.

177. **First**, the broadest reasonable interpretation of the “electronically simulate a physical disconnection and reconnection” claim language covers the “reset” operation in Michelson, as discussed above in Section X.C.4. The reset line 63 is part of a reset circuit. As noted above in Section X.C.4, the ’770 Patent specifies in claim 10 that the second circuit configured to electronically simulate a disconnect and reconnect over the bus can be a reset circuit. The “reset” of the FPGA 60 is an example of an “electronic reset” because FPGA is an electronic circuit and operates in response to an electrical reset signal. Ex. 1003 4:17–22. The “reset” of Michelson “electronically simulat[es] a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device to a second configuration based on the second set of configuration information” because a “reset” is associated with physical disconnection and reconnection of a PCMCIA card over a bus, Ex. 1008, pp. 4-6 to 4-7 and 4-10 to 4-12, and the reset reconfigures the FPGA 60 to the configuration information downloaded over the PCMCIA bus, *id.* at pp. 4-10 to 4-11, 5-20; Ex. 1019, pp. 3-20 to 3-24; Ex. 1020, 2:51–3:4; 5:66–6:9; Ex. 1003, 4:17–22. Also, the reset line 63 in Michelson is part of a reset circuit, and the ’770 Patent specifies that the second circuit can be a reset circuit. Thus, the disclosure of a reset circuit in Michelson teaches a circuit configured to electronically simulate a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device to a second configuration.

178. **Second**, even if the above limitation is construed more narrowly to include a switch, which it should not be, PCCextend teaches that it was known in the prior art to provide a switch to simulate a card removal (disconnect) and insertion (reconnect) cycle for PCMCIA cards. PCCextend describes a “PCMCIA extender card” which is inserted between a PC card under test and a socket in the host system. Ex. 1004, p. 1. PCCextend states: “Caution: Insertion and removal of the extender and PC card should be done with care. The PC Card’s fragile connectors may be broken or bent if improper force is used.” *Id.*, at p. 1. PCCextend describes that the extender card has a PCCswitch SW1, where “the PCCswitch can interrupt the card detect signals (-CD 1 and -CD2) to simulate a card removal/insertion cycle.” *Id.* at p. 3. Using the PCCSwitch to interrupt the card detect signals on the bus in PCCextend is an example of electronically simulating a disconnect and reconnect. It would have been obvious to a person of ordinary skill in the art to modify the card of Michelson to include a switch on the card detect lines as taught by PCCextend to reprogram the card because it was well-known that PCMCIA cards could be reprogrammed by removing and reinserting the card, and PCCextend teaches that the PCCswitch can be used to simulate such a removal/insertion cycle. Ex. 1003, 1:46–49; Ex. 1004, p. 3.

179. Further, it would have been obvious to a person of ordinary skill in the art at the time of the invention to apply the use of the PCCswitch in PCCextend to

the reprogramming operation in Michelson to avoid the need to physically disconnect and reconnect the card in view of the fragile nature of the connectors. Simulating a card removal/insertion cycle using the PCCSwitch causes the peripheral device to be reconfigured. Ex. 1017, pp. 119–123; Ex. 1018, 4-10 to 4-11; 5-21. *See also* Ex. 1019, p. 3–21.

180. It was well known that PCMCIA cards could be reprogrammed by removing and re-inserting the card. Simulating a card removal/insertion cycle using the PCCSwitch causes the peripheral device to be reconfigured. *Id.*; *see also* Ex. 1017, pp. 119–123; Ex. 1018, 4-10 to 4-11; 5-21. *See also* Ex. 1019, p. 3–21. It would have been obvious to a person of ordinary skill in the art to modify the card of Michelson to include a switch on the card detect lines as taught by PCCextend to reprogram the card because it was well-known that PCMCIA cards could be reprogrammed by removing and reinserting the card, and PCCextend teaches that the PCCswitch can be used to simulate such a removal/insertion cycle. Ex. 1003, 1:46–49 and Ex. 1004, p. 3.

181. To the extent the term may be more narrowly construed to require an electronic switch, which it should not, Davis discloses an electronic switch in the card detect line that electronically simulates a physical disconnection and reconnection of a peripheral device over a PCMCIA bus. Davis states that “the **switching device 23** can be implemented by an electronic switch, typically a field

effect transistor (FET) or a bipolar transistor.” Ex. 1005, 7:31–34. Davis states that “a device removal event can be represented by deactivating the FET and opening this single<sup>7</sup>] path. Likewise, a device insertion event can be represented by activating the FET and closing this signal path.” Ex. 1005, 10:29–32. Interrupting the card detect signals on the bus using the FET switch of Davis is another example of simulating a disconnect and reconnect over a bus. It would have been obvious to use an electronic switch in the combination of Michelson and PCCextend to simulate a disconnect and reconnect as taught by Davis to avoid the need for physically disconnecting and reconnecting the device or pushing a button to activate a switch.

182. Moreover, one of ordinary skill in the art would have been motivated to substitute the FET switch in Davis for the manual switch in PCCextend, as choosing the type of switch to simulate a physical disconnection would have been merely a matter of design choice. In either case, the simulated disconnection/reconnection would cause the device to be reconfigured.

183. Claim 1 further recites that the second circuit is configured to electronically simulate the physical disconnection and reconnection “while supplying electrical power to said peripheral device.”

---

<sup>7</sup> One of ordinary skill in the art, reading the specification, would have recognized that “single” contains a typographical error and should read “signal.”



184. Both Michelson and PCCextend teach that the electronic simulation of a physical disconnection and reconnection is performed while electrical power is supplied to the peripheral device.

185. As noted above, the reset in Michelson meets the electronic simulation of a physical disconnection/reconnection limitation in claim 1. In Michelson, a circuit 68 in the peripheral device drives the reset line 63 to reprogram the device. Ex. 1003, 4:17–22; 5:45–51; and FIG. 3. It was well-known at the time of Michelson for a host computer to power a PCMCIA peripheral device over the PCMCIA bus. Since the reset process disclosed in Michelson occurs in the peripheral device, it does not affect the supply of power to the device over the PCMCIA bus. In fact, the device in Michelson must remain powered by the bus in order to perform the reset process. *Id.* Thus, Michelson teaches electronic simulation of a physical disconnection/reconnection while supplying electrical power to the peripheral device.

186. PCCextend indicates that power is supplied to the PCCextend100 board, which is part of the peripheral device, through VCC supply pins. Ex. 1004 at 3. For instance, a power LED on the board lights up when “a Vcc of greater than 3.5V is present.” *Id.* Further, PCCextend shows that the Vcc pins are not affected by the PCCSwitch SW1. *Id.* at pp. 15–16. PCCextend advises that “[t]he software may also remove power from the socket when the card is removed,” *Id.* at 5

(emphasis added), which one of ordinary skill in the art would interpret to mean that there are circumstances where power is not removed from the socket and, thus, supplied to the device when the PCCSwitch is used to simulate a physical disconnection and reconnection. As such, the proposed combination of Michelson, PCCextend, and Davis would include this feature. Alternatively, it would have been obvious for the proposed combination to include this feature so that the peripheral device could perform a reset or other functions requiring power.

187. Thus, it is my opinion that claim 1 would have been obvious over Michelson, PCCextend, and Davis.

## **2. Dependent Claim 2**

188. Claim 2 depends from independent claim 1, which is addressed above in Section X.E.1. With respect to the additional limitations set forth in claim 2, as noted in Section X.D.1, Michelson teaches the limitation that “said first configuration is a generic configuration assigned to the peripheral device and said second configuration comprises any one of a plurality of unique manufacturer configurations.”

189. As such, the combination of Michelson, PCCextend, and Davis would include this feature. Thus, claim 2 would have been obvious over the combination of Michelson, PCCextend, and Davis.

### **3. Dependent Claim 3**

190. Claim 3 depends from claim 2, which is addressed in Section X.E.2, and further recites that “said first circuit is configured to (i) read an identification code from the peripheral device and (ii) select said second configuration based on said identification code.” As noted above in Section X.D.2, Michelson discloses this feature, and/or it would have been obvious in view of Michelson. As such, the combination of Michelson, PCCextend, and Davis would include this feature.

191. Thus, it is my opinion that claim 3 would have been obvious over Michelson, PCCextend, and Davis.

### **4. Dependent Claim 10**

192. Claim 10 depends from independent claim 1, which is addressed above in Section X.E.1. With respect to the additional limitations set forth in claim 10 that “wherein said second circuit comprises a reset circuit configured to reset the first or second configuration of the peripheral device,” this limitation is taught by Michelson for the reasons discussed above in Section X.E.1 with respect to claim 1. Also, it was well known at the time that a card removal and insertion event as simulated by PCCextend and Davis would result in a reset. Ex. 1018, pp. 4-6 to 4-7; *see also id.* at p. 4-11. Further, it was well known that resetting a PCMCIA device included resetting a configuration of the device (*i.e.*, a first or second configuration). *See, e.g.*, Ex. 1017, pp. 119–126; Ex. 1018, pp. 4-6 to 4-7, 4-10 to

4-11, 5-21; Ex. 1019, pp. 3-14 to 3-16, 3-20 to 3-24, 3-28 to 3-29, 5-79, B-14; Ex. 1020, 2:51–3:4; 5:66–6:9. As such, the combination of Michelson, PCCextend, and Davis would include this feature.

193. Thus, claim 10 would have been obvious over Michelson, PCCextend, and Davis.

### **5. Independent Claim 11**

194. Claim 11 is a method claim version of system claim 1. That is, claim 11 recites method steps performed by the first and second circuits in claim 1 and recited in that claim. Thus, claim 11 would have been obvious over Michelson, PCCextend, and Davis for the same reasons as claim 1, as discussed in Section X.E.1 above.

### **6. Dependent Claim 12**

195. Claim 12 depends from independent claim 11, which is addressed above in Section X.E.5. Claim 12 further recites “wherein said first configuration comprises a generic configuration assigned to the peripheral device and said second configuration comprises any one of a plurality of unique manufacturer configurations.” Thus, claim 12 is a method version of system claim 2. As discussed above with respect to claim 2 in Section X.E.2, this feature is disclosed by Michelson, and/or it would have been obvious in view of Michelson. Ex. 1003, 3:34–4:8. Thus, claim 12 would have been obvious over Michelson, PCCextend,

and Davis for at least the same reasons discussed above with respect to claim 2 in Section X.E.2.

### **7. Dependent Claim 13**

196. Claim 13 depends from claim 12, which is addressed above in Section X.E.6. Claim 13 further recites “wherein step (A) comprises: reading an identification code from the peripheral device, and selecting said second configuration based on said identification code.” Thus, claim 13 is a method version of system claim 3. As discussed above with respect to claim 3 in Section X.E.3, this feature is disclosed by Michelson and/or would have been obvious in view of Michelson. Thus, claim 13 would have been obvious over Michelson, PCCextend, and Davis for the same reasons discussed above with respect to claim 3 in Section X.E.3.

### **8. Dependent Claim 16**

197. Claim 16 depends from independent claim 11, which is addressed above in Section X.E.5. Claim 16 further recites “wherein said information for the second configuration comprises (i) configuration data and (ii) executable code.” Michelson discloses this feature.

198. Michelson discloses downloading a FPGA programming data file into the card controller FPGA 60 of the PCMCIA device. *E.g.*, Ex. 1003, 3:59–4:22;

5:36–50. Michelson discloses that the FPGA programming data file controls the functionality of the card controller FPGA 60. *Id.* at 5:51–6:62.

199. One of ordinary skill in the art would understand that the FPGA programming data file disclosed in Michelson is a combination of both “configuration data” (because it includes data) and “executable code” (because it programs the FPGA for execution). It would also have been obvious to one of ordinary skill in the art to include additional information, such as a version number, in the FPGA programming data file to allow the host to determine the device characteristics. Furthermore, it would have been obvious to one of ordinary skill in the art to program the peripheral device in Michelson with microprocessor-executable software or firmware containing data and code, instead of with FPGA programming data file, because these were both well-known ways of programming a peripheral device at the time and were known substitutes that yielded predictable results. *See, e.g.,* Snyder (Ex. 1015), 9:18–20 (“Although in the preferred embodiment the USB microcontroller 8 includes several FPGAs, RAM and EEPROMs ... the invention may be implemented using a conventional general purpose digital computer or microprocessor programmed according to the teachings ... [and a]ppropriate software coding can readily be prepared.”). Moreover, the APA of the ’770 Patent admits that it was known in the prior art “to alter the configuration or personality of a peripheral device, such as downloading **new code**

**or configuration information** into the memory of the peripheral device.” Ex. 1001, 2:24–28. It would have been obvious to include one or both of the new code and configuration information, depending on the type of configuration change desired. *See also* Bedingfield (Ex. 1020), 2:6–66; 5:42–55; 4:27–3:33; 7:28–33. As such, it would have been obvious to include the feature in the combination of Michelson, PCCextend, and Davis. Thus, claim 16 would have been obvious over the combination of Michelson, PCCextend, and Davis.

### **9. Dependent Claim 17**

200. Claim 17 depends from independent claim 11, which is addressed above in Section X.E.5. With respect to the additional feature set forth in claim 17 that “wherein step (B) comprises electronically resetting the configuration of the peripheral device,” Michelson discloses and/or suggests this feature, as discussed above in Section X.E.4.

201. Claim 17 additionally recites that the resetting is “controllable by the peripheral device.” Michelson discloses this feature as well. More specifically, Michelson discloses that after the processor downloads FPGA programming data file 42 through PCMCIA adapter 16 to PCMCIA interface chip 26, the FPGA is “reset” by FPGA programming circuit 68 to enable reprogramming:

Referring to FIG. 3, card controller 32 includes a PCMCIA card controller FPGA 60 (e.g., part number XC3042TQ100-100, manufactured by Xilinx, as described in Xilinx Programmable Logic

Data Book, which is hereby incorporated by reference). . . . Interface chip 26 initiates FPGA 60 programming through FPGA programming circuit 68, which drives reset line 63 and reprogram line 65, and completes FPGA 60 programming by loading the FPGA programming data into FPGA 60 through peripheral data lines 72.

Ex. 1003, 4:9–22.

202. The configuration of the card controller FPGA 60 is electronically “reset” by a reset signal on line 63 from the programming circuit 68 in response to the interface chip 26. *Id.* Programming circuit 68 is part of the PCMCIA card 14. *See, e.g.*, Ex. 1003, Fig. 3, 4:9–22. Thus, the “reset” is “controllable by the peripheral device.”

203. Further, PCCextend teaches a PCCSwitch which also resets the configuration by simulating a device removal and attachment. Ex. 1004, pp. 1, 3, and 4. The PCCSwitch is part of a circuit on the peripheral device, and the reset is thus controllable by the peripheral device. Further, one of ordinary skill in the art would have recognized that in this context, whether a switch is controlled by the host or by the peripheral device is purely a matter of design choice and that controlling the switch from the peripheral device provides no new and unexpected results.

204. For at least these reasons, the combination of Michelson, PCCextend, and Davis would include this feature. Thus, claim 17 would have been obvious over the combination of Michelson, PCCextend, and Davis.



## 10. Independent Claim 18

205. Claim 18 recites “A system for reconfiguring a peripheral device having a configuration connected by a computer bus to a host computer.”

Michelson “relates to programming and reprogramming the hardware configuration of a (PCMCIA) card.” Ex. 1003, 1:7–16. A PCMCIA card is “typically used to add functionality or memory to a personal, portable, or desktop computer (i.e., a host computer),” *id.* at 1:13–15, and is thus a type of peripheral device. The system for “[r]eprogramming the hardware configuration” teaches a system for “reconfiguring” a peripheral device because it changes the configuration of the card. Thus, Michelson teaches a system for reconfiguring a peripheral device.

206. Michelson states that a “typical PCMCIA card includes a standard PCMCIA connector connected to a PCMCIA interface circuit through a standard PCMCIA bus.” (Ex. 1003, 1:28–30) and “the host computer includes a PCMCIA adapter circuit coupled to a PCMCIA host socket which is mechanically and electrically connected to a PCMCIA card connector on the PCMCIA card” (Ex. 1003, 2:17–20), and “PCMCIA card 14 card connector 28 is inserted in PCMCIA host socket 18 of host computer 12,” *id.* at 3:34–37. “Through a standard PCMCIA bus (*i.e.*, PCMCIA address lines 62, data lines 64, and control lines 66) connected to PCMCIA connector 28”, the peripheral PCMCIA device receives data from the host computer. *Id.* at 4:13–23. Fig. 1 of Michelson shows a PCMCIA card connected

to a socket of a host computer. Fig. 3 shows a PCMCIA card having a PCMCIA bus (lines 62, 64, 66) connected to the connector 28. The PCMCIA bus is an example of a computer bus. Thus, Michelson discloses a peripheral device “connected by a computer bus to a host computer.”

207. Michelson discloses that the PCMCIA card has a first configuration, such as a generic configuration that is defined at least in part by the Card Information Structure (CIS) data, which is read by the host computer to configure the computer and the card to operate together enable the host processor 22 to select the appropriate application software 40 from the host memory 24. Ex. 1003, 3:34–58. Thus, Michelson teaches that the peripheral device has “a configuration.”

208. Claim 18 further recites a “first circuit configured to detect the peripheral device connected to the computer bus.” Michelson discloses this feature.

209. For example, Michelson discloses that, “when PCMCIA card 14 card connector 28 is inserted in PCMCIA host socket 18 of host computer 12, PCMCIA adapter 16 recognizes (step 44) the insertion . . . .” Ex. 1003, 3:34–37.

210. Claim 18 further recites a “second circuit configured to electronically simulate a physical disconnection and reconnection of the peripheral device to reset said configuration of said peripheral device.”

211. As noted above in Section X.E.1, Michelson discloses that, after the processor downloads FPGA programming data file 42 through PCMCIA adapter 16 to PCMCIA interface chip 26, the FPGA is “reset” to enable reprogramming. Ex. 1003, 4:17–22. The electronic reset of Michelson is an example of “electronically simulating a physical disconnection and reconnection of the peripheral device,” as discussed above in Section X.E.1. The electronic reset resets the configuration of the peripheral device, because it is part of the reprogramming of the peripheral device.

212. Thus, Michelson teaches the feature of “a second circuit configured to electronically simulate a physical disconnection and reconnection of the peripheral device to reset said configuration of said peripheral device.”

213. Additionally, even if the above limitation is construed more narrowly, which it should not be, PCCextend teaches that it was known in the prior art to provide a switch to simulate a card removal (disconnect) and insertion (reconnect) cycle for PCMCIA cards to reprogram the card. Ex. 1003, pp. 3–4 (Figure 2.3-1 omitted). PCCextend describes a “PCMCIA extender card” which is inserted between a PC card under test and a socket in the host system. Ex. 1004, p. 1. PCCextend states: “Caution: Insertion and removal of the extender and PC card should be done with care. The PC Card’s fragile connectors may be broken or bent if improper force is used.” Ex. 1004, p. 1. PCCextend describes that the extender

card has a PCCswitch SW1, where “the PCCswitch can interrupt the card detect signals (-CD 1 and -CD2) to simulate a card removal/insertion cycle.” *Id.* at p. 3. Using the PCCSwitch to interrupt the card detect signals on the bus in PCCextend is an example of electronically simulating a disconnect and reconnect.

214. As noted above in Section X.E.1, it would have been obvious to a person of ordinary skill in the art at the time of the invention to apply the use of the PCCswitch in PCCextend the reprogramming operation in Michelson, such as to avoid the need to physically disconnect and reconnect the card in view of the fragile nature of the connectors. Simulating a card removal/insertion cycle using the PCCSwitch causes the peripheral device to be reconfigured. *Id.*; see also Ex. 1017, pp. 119–123; Ex. 1018, 4-10 to 4-11; 5-21. See also Ex. 1019, p. 3-21.

215. Furthermore, as noted above in Section X.E.1, to the extent the electronically simulate term may be construed even more narrowly to require an electronic switch in the card detect line that electronically simulates a physical disconnection and reconnection over a PCMCIA bus, Davis discloses a switch in the form of a FET that electronically simulates a physical disconnection and reconnection of a peripheral device. Ex. 1005, 7:31–34; 8:31–34.

216. Davis discloses controlling the opening and closing of the switch with the power management module to simulate a disconnection (removal) and reconnection (insertion). Ex. 1005, 9:41–52 and 10:1–20. Davis states that “a

device removal event can be represented by deactivating the FET and opening this single path. Likewise, a device insertion event can be represented by activating the FET and closing this signal path.” *Id.* at 10:29–32. Interrupting the card detect signal on the bus using the FET switch of Davis is another example of simulating a disconnect and reconnect. It would have been obvious to use an electronic switch in the combination of Michelson and PCCextend to avoid the need for physically disconnecting and reconnecting the device or pushing a button to activate a switch.

217. It was well known at the time that removal/insertion of a PCMCIA card would result in a reset of the card. (Ex. 1017, pp. 119–123; Ex. 1018, 4-10 to 4-11; 5-21. *See also* Ex. 1019, p. 3-21). Thus, modifying Michelson to include card detect line switches as taught by PCCextend and Davis would cause the configuration of the card to be reset.

218. As noted above in Section X.E.1, one of ordinary skill in the art would have been motivated to substitute the FET switch in Davis for the manual switch in PCCextend, as choosing the type of switch to simulate a physical disconnection and reconnection would have been merely a matter of design choice. In either case, the simulated disconnection/reconnection would cause the configuration of the device to be reset.

219. Claim 18 further recites that the second circuit is configured to electronically simulate the physical disconnection and reconnection “while supplying electrical power to said peripheral device.”

220. As noted above in Section X.E.1, Michelson discloses electronically simulating a physical disconnection/reconnection while supplying power to the peripheral device.

221. Further, as stated above in Section X.E.1, PCCextend discloses simulating a physical disconnection and reconnection of the peripheral device by changing the voltage on a card detect line without removing power to the peripheral device.

222. As such, it would have been obvious to one of ordinary skill in the art to combine Michelson, PCCextend, and Davis to perform the electronic simulation of a disconnection and reconnection of a peripheral device “while supplying electrical power to said peripheral device” for the same reasons discussed above in Section X.E.1 in connection with claim 1.

223. Thus, it is my opinion that claim 18 would have been obvious over Michelson, PCCextend, and Davis.

#### **11. Dependent Claim 20**

224. Claim 20 depends from independent claim 18, which is addressed above. With respect to the additional limitation set forth in claim 20 that “said

second circuit comprises a solid state switch,” Davis teaches that the second circuit configured to electronically simulate a physical disconnection and reconnection comprises a solid state switch (*e.g.*, a FET). Ex. 1005, 7:23–35; 10:21–39; Fig. 3.

225. It would have been obvious to use a solid state switch in the combination of Michelson and PCCextend to electrically simulate a physical disconnection and reconnection of a PCMCIA card in view of Davis as doing so would merely be applying a well-known prior art switching component in its intended role (*i.e.*, as a switch that simulates a physical disconnection) to yield predictable results. Thus, claim 20 would have been obvious over Michelson, PCCextend, and Davis.

**F. Claims 5, 7, 15, 19 are unpatentable under 35 U.S.C. § 103(a) as being obvious over Michelson in view of PCCextend, Davis, and the APA**

**1. Dependent Claim 5**

226. Claim 5 depends on claim 1, which is addressed above in Section X.E.1, and further recites “wherein said computer bus comprises a Universal Serial Bus.”

227. With respect to the additional limitations set forth in claim 5, Michelson describes communicating a second set of configuration information to the peripheral device over a PCMCIA bus (*e.g.*, Ex. 1003, 4:13–17), not a USB.

228. However, as described in the APA, the advantages of the USB were well known:

A new emerging technology called the Universal Serial Bus (USB) is a system intended to create a single standardized peripheral device connection system. The USB makes the task of connecting peripheral devices to computers easier and more reliable since it uses a standardized connector and form factor, and makes operating those peripheral devices with the computer, easier and more reliable than with the various different types of communication ports. The computer to which these peripheral devices are connected by the USB is known as the “host computer”. The USB replaces the multiple cable and connector types with a single standardized connection system. The USB also permits the connection and disconnection of USB compatible peripheral devices while the computer is turned on which eliminates the typical turning off and rebooting of the computer in order to connect or disconnect a peripheral device to the computer.

Ex. 1001, 1:50–65.

229. One of ordinary skill in the art of interfacing peripheral devices would have been motivated to substitute a USB for a PCMCIA bus to achieve the known advantages of the more modern USB. Ex. 1001, 1:50–64. Additionally, the substitution would have involved only routine engineering, *see, e.g.*, Snyder (Ex. 1015) at 9:31–34, particularly in view of the similarities between PCMCIA and USB. For example, both monitor bus lines to detect disconnections and



reconnections, both reconfigure a device when it is connected, and both include a reset as part of the reconfiguration. *See, e.g.*, Ex. 1001, 1:66–2:21; 4:15–62; 6:17–44; Figs. 1 and 3; Ex. 1013, pp. 116–119, 169; Ex. 1005, 6:37–54; Ex. 1017, pp. 119–126; Ex. 1018, pp. 4-6 to 4-7, 4-10 to 4-11, 5-21; Ex. 1019, pp. 3-14 to 3-16, 3-20 to 3-24, 3-28 to 3-29, 5-79, B-14; Ex. 1020, 2:51–3:4; 5:66–6:9. Thus, it would have been obvious to one of ordinary skill in the art to communicate the information for the second configuration to the peripheral device over a USB. Furthermore, although not required in claim 5, it would also have been obvious to electronically simulate a physical disconnection and reconnection over the USB by incorporating a switch as taught in PCCextend/Davis on the bus lines (D+ and D-) of a USB, since these lines are also used for device detection, and the resulting combination would allow simulation of a disconnection and reconnection of the peripheral device to reconfigure the device to a second configuration based on the received configuration information. Furthermore, simulating a disconnect and reconnect in the proposed combination (*e.g.*, by placing switches on D+ and/or D- lines) would result in a reconfiguration of the device to a second configuration while supplying power to the peripheral device (*e.g.*, using Vbus and GND lines). Ex. 1013, pp. 29–30, 116–119, 169.

230. Thus, claim 5 would have been obvious over the combination of Michelson, PCCextend, Davis, and the APA.

## **2. Dependent Claim 7**

231. Claim 7 depends on claim 5, which is addressed above in Section X.F.1, and further recites “wherein said information for said second configuration comprises (i) configuration data and (ii) an executable code.” Thus, claim 7 is a system version of method claim 16. As noted above in Section X.E.8, both Michelson and the APA teach or suggest this feature, for the reasons discussed above in Section X.E.8.

232. Thus, claim 7 would have been obvious over Michelson, PCCextend, Davis, and the APA.

## **3. Dependent Claim 15**

233. Claim 15 depends on claim 11, which is addressed above in Section X.E.5, and further recites “wherein step (A) comprises communicating said information for the second configuration to the peripheral device using a Universal Serial Bus and port.”

234. Michelson teaches or suggests downloading by communicating a second set of configuration information to the peripheral device over a PCMCIA bus (*e.g.*, Ex. 1003, 4:13–17), not a USB. However, as described in the APA, the advantages of the USB were well known:

A new emerging technology called the Universal Serial Bus (USB) is a system intended to create a single standardized peripheral device connection system. The USB makes the task of connecting

peripheral devices to computers easier and more reliable since it uses a standardized connector and form factor, and makes operating those peripheral devices with the computer, easier and more reliable than with the various different types of communication ports. The computer to which these peripheral devices are connected by the USB is known as the “host computer”. The USB replaces the multiple cable and connector types with a single standardized connection system. The USB also permits the connection and disconnection of USB compatible peripheral devices while the computer is turned on which eliminates the typical turning off and rebooting of the computer in order to connect or disconnect a peripheral device to the computer.

Ex. 1001, 1:50–65.

235. One of ordinary skill in the art of interfacing peripheral devices would have been motivated to substitute a USB for a PCMCIA bus to achieve the known advantages of the more modern USB. Ex. 1001, 1:50–64. Additionally, the substitution would have involved only routine engineering, *see, e.g.*, Snyder (Ex. 1015) at 9:31–34, particularly in view of the similarities between PCMCIA and USB. For example, both monitor bus lines to detect disconnections and reconnections, both reconfigure a device when it is connected, and both include a reset as part of the reconfiguration. *See, e.g.*, Ex. 1001, 1:66–2:21; 4:27–62; 6:17–44; Figs. 1 and 3; Ex. 1013, pp. 116–119, 169; Ex. 1005, 6:37–54; Ex. 1017, pp. 119–126; Ex. 1018, pp. 4-6 to 4-7, 4-10 to 4-11, 5-21; Ex. 1019 at pp. 3-14 to 3-16,

3-20 to 3-24, 3-28 to 3-29, 5-79, B-14; Ex. 1020, 2:51–3:4; 5:66–6:9. Thus, it would have been obvious to one of ordinary skill in the art to communicate the information for the second configuration to the peripheral device over a USB. Furthermore, although not required in claim 15, it would also have been obvious to electronically simulate a physical disconnection and reconnection over the USB by incorporating a switch as taught in PCCextend/Davis on the bus lines (D+ and D-) of a USB, since these lines are also used for device detection, and the resulting combination would allow simulation of a disconnection and reconnection of the peripheral device to reconfigure the device to a second configuration based on the received configuration information while supplying power to the peripheral device (e.g., using Vbus and GND lines). Ex. 1013, pp. 29–30, 116–119, 169. Thus, claim 15 would have been obvious over Michelson, PCCextend, Davis, and the APA.

#### **4. Dependent Claim 19**

236. Dependent claim 19 depends from independent claim 18, which is addressed above in Section X.E.10, and further recites “where said computer bus comprises a Universal Serial Bus.” It would have been obvious to modify Michelson, PCCextend and Davis to include a Universal Serial Bus. That is, one of ordinary skill in the art of interfacing peripheral devices would have been motivated to substitute a USB for a PCMCIA bus to achieve the known advantages of the more

modern USB, and the substitution would have involved only routine engineering, particularly in view of the similarities between PCMCIA and USB. *See also supra* Section X.F.1. Furthermore, simulating a disconnect and reconnect in the proposed combination (e.g., by placing switches on D+ and/or D- lines) would result in a reset of the configuration of the device while supplying power to the peripheral device (e.g., using Vbus and GND lines). Ex. 1013, pp. 29–30, 116–119, 169.

237. Thus, claim 19 would have been obvious over Michelson, PCCextend, Davis, and the APA.

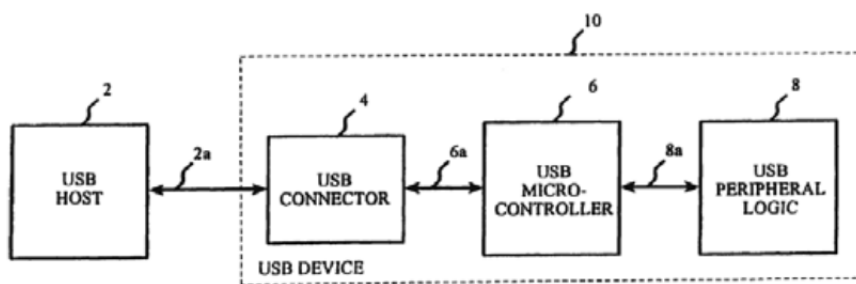
**G. Claims 18–20 are unpatentable under 35 U.S.C. § 102(e) as being anticipated by Yap**

**1. Independent Claim 18**

238. Independent claim 18 recites in the preamble a “system for reconfiguring a peripheral device having a configuration connected by a computer bus to a host computer.”

239. Yap teaches a system that simulates a physical removal of a USB device (*i.e.*, a peripheral device) to “allow the USB host to re-configure the USB device.” Ex. 1002, 4:6–10. Thus, the peripheral device has a configuration. The peripheral device in Yap is connected by a “universal serial bus (‘USB’)” to a host computer. Ex. 1002, Fig. 1 (reproduced below); 3:50–59. Yap further incorporates by reference the USB Specification v1.0, which teaches a computer bus and port for connecting USB devices, like the one in Yap, to a host computer. Ex.

1002, 1:38–42; Ex. 1013, pp. 19 (“For Universal Serial Bus, the [port is the] point where a Universal Serial Bus device is attached.”); 28 (“The Universal Serial Bus connects USB devices with the USB host.”). As such, Yap discloses a system for simulating a disconnection and reconnection of a peripheral device connected by a computer bus and a port to a host computer. Thus, Yap discloses all the features of the preamble.



*Fig. 1*

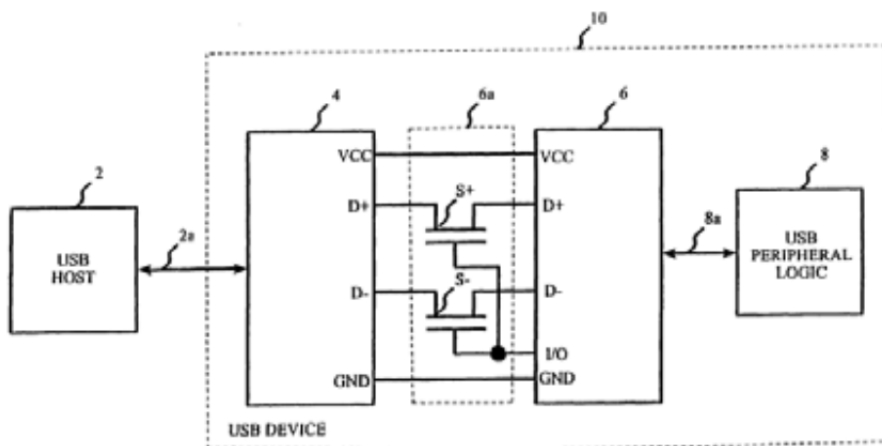
240. Claim 18 further recites “a first circuit configured to detect the peripheral device connected to the computer bus.”

241. Yap teaches a system that emulates the disconnection of a USB device from the USB host and the re-connection of the USB device to the USB host. Ex. 1002, 4:19–23; 5:65–6:3. When a USB peripheral device is connected or re-connected to the USB host, the USB host necessarily detects the peripheral device in order to be able to interact with the peripheral device. Further, the USB v 1.0 Specification, which is incorporated by reference in Yap, teaches that USB hosts

have circuits that are configured to detect the peripheral device connected to the port by monitoring the voltage on the D+ and D- data lines. Ex. 1013 (USB v1.0 Specification), p. 112. Thus, the USB host computer in Yap requires a circuit configured to detect a peripheral device connected to a port.

242. Claim 18 further recites “a second circuit configured to electronically simulate a physical disconnection and reconnection of the peripheral device to reset said configuration of said peripheral device.”

243. Yap teaches a circuit that is configured to electronically simulate the physical disconnection and reconnection of the peripheral device over the computer bus using switching devices S+ and S- in the data lines D+ and D-, respectively, as shown in Figure 2:



*Fig. 2*

244. Yap states that “[b]y disconnecting the D+ and D- data lines via switching devices S+ and S-, a physical removal of the USB device 10 may be

simulated in order to allow the USB host to re-configure the USB device 10 . . . .”

Ex. 1002, 4:6–9. Yap further states: “[T]he USB micro-controller 6 opens the data lines via the switching devices S+ and S- for a duration greater than 2.5 microseconds and then reconnects them again. This procedure, for example, emulates the disconnect and re-connect procedure as specified in the USB specification v1.0, page 116.” Ex. 1002, 4:18–13. Figure 3 of Yap discloses a second embodiment with the second circuit and switching devices S+ and S- within the USB microcontroller. Ex. 1002, 4:24–38. As such, Yap discloses a circuit that is configured to electronically simulate the physical disconnection and reconnection of the peripheral device over the computer bus.

245. The simulated disconnection and reconnection in Yap resets the configuration of the peripheral device. *See supra* Section X.C.4.

246. Claim 18 further recites that the second circuit is configured to electronically simulate the physical disconnection and reconnection of the peripheral device “while supplying electrical power to said peripheral device.”

247. Yap teaches that the electronic simulation of a physical disconnection and reconnection is performed while electrical power is supplied to the peripheral device. For a more detailed discussion, see above with respect to claim 1 in Section X.C.1.

248. Thus, in my opinion, claim 18 is anticipated by Yap.



## **2. Dependent Claim 19**

249. Claim 19 depends from independent claim 18, which is addressed above in Section X.G.1. With respect to the additional limitation set forth in claim 19 that “wherein said computer bus comprises a Universal Serial Bus,” Yap teaches that the computer bus and port comprise a “Universal Serial Bus (‘USB’).” Ex. 1002, Abstract; 3:50–59; Fig. 1. Thus, claim 19 is anticipated by Yap.

## **3. Dependent Claim 20**

250. Claim 20 depends from independent claim 18, which is addressed above in Section X.G.1. With respect to the additional limitations set forth in claim 20 that “wherein said second circuit comprises a solid state switch,” Yap teaches that the switches S+ and S- may be transistors. Ex. 1002, 3:60–63; claims 14 and 15. Transistors are solid state switches. Further, Figs. 2 and 3 show that the switches S+ and S- are solid state FET transistors. Thus, claim 20 is anticipated by Yap.

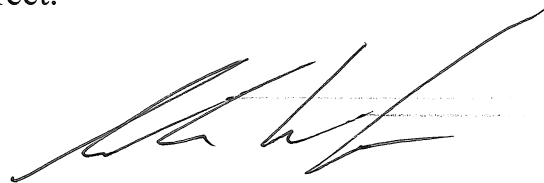
## **XI. CONCLUDING STATEMENTS**

251. In signing this declaration, I understand that the declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I acknowledge that I may be subject to cross-examination in the case and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

252. I declare that all statements made herein of my knowledge are true and that all statements made on information and belief are believed to be true; and further, that these statements were made with knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. § 1001.

253. I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct.

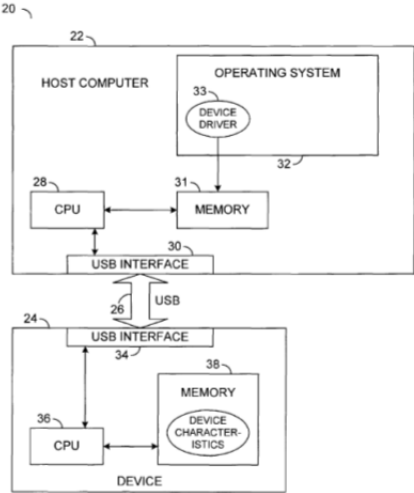
Date: September 12, 2014

A handwritten signature in black ink, appearing to read 'Andrew Wolfe', written in a cursive style.

Andrew Wolfe, Ph.D.

# Appendix A

**A. Ground 1: Claims 1, 5, 7, 10, 11, and 15-17 are unpatentable under 35 U.S.C. § 103, as being obvious over the Admitted Prior Art (APA) and U.S. Patent No. 6,073,193 to Yap**

U.S. Patent 6,593,770	Correspondence to Admitted Prior Art (Ex. 1001) and Yap (Ex. 1002)
<b>Claim 1</b>	
<p>[a] A system for reconfiguring a peripheral device having a first configuration connected by a computer bus to a host computer, the system comprising:</p>	<p>APA, Ex. 1 00 1, Prior Art Fig. 1:</p>  <p style="text-align: center;"><b>FIG. 1</b> (PRIOR ART)</p> <p>APA, Ex. 1001, 4:15-34:</p> <p>FIG. 1 is a diagram illustrating a standardized bus interface, such as a conventional computer system 20, that may include a host computer system 22 and a peripheral device 24. The peripheral device is connected to the host computer by a universal serial bus (USB) 26. The host computer may include a central processing unit (CPU) 28 connected to a USB interface (I/F) circuit 30, and the USB standard provides a universal electrical and physical interface for the peripheral devices via bus 26. The CPU executes software application code located in a memory 31 and communicates data to and from the peripheral device through the USB interface and the USB 26. The host computer may also include an operating system 32 which may include a software device driver 33. The peripheral device 24 may include a USB interface circuit 34, a CPU 36</p>

U.S. Patent 6,593,770	Correspondence to Admitted Prior Art (Ex. 1001) and Yap (Ex. 1002)
	<p>and a non-volatile memory 38 that may store configuration information describing the characteristics of the peripheral device. The non-volatile memory may be a read only memory (ROM) or an erasable programmable read only memory (EPROM).</p> <p>APA, Ex. 1001, 1:50-2:19:</p> <p>A new emerging technology called the Universal Serial Bus (USB) is a system intended to create a single standardized peripheral device connection system. The USB makes the task of connecting peripheral devices to computers easier and more reliable since it uses a standardized connector and form factor, and makes operating those peripheral devices with the computer, easier and more reliable than with the various different types of communication ports. The computer to which these peripheral devices are connected by the USB is known as the “host computer”. The USB replaces the multiple cable and connector types with a single standardized connection system. The USB also permits the connection and disconnection of USB compatible peripheral devices while the computer is turned on which eliminates the typical turning off and rebooting of the computer in order to connect or disconnect a peripheral device to the computer.</p> <p>When a peripheral device is first connected to the USB and the host computer through a standard USB communications port, the presence of the connected peripheral device is detected and a configuration process of the USB for the connected peripheral device, known as device enumeration, begins. The enumeration process assigns a unique USB address to the connected peripheral device, queries the connected peripheral device about its requirements and capabilities, writes data about the connected peripheral device into the host computer’s operating system, and loads the appropriate software device driver from a storage location into the host</p>

U.S. Patent 6,593,770	Correspondence to Admitted Prior Art (Ex. 1001) and Yap (Ex. 1002)
	<p>computer's operating system. During the query, a data table stored in the peripheral device, which contains the particular peripheral device's configuration information, is read from the peripheral device into the host computer's memory. Upon completion of the enumeration process, the connected peripheral device is recognized by the host computer's operating system and may be used by application software being executed by the microprocessor of the host computer. The association of the device with the software device driver cannot be subsequently changed.</p> <p>APA, Ex. 1001, 2:20-39:</p> <p style="padding-left: 40px;">In a serial bus system, such as the USB, the only opportunity for associating software device drivers with a peripheral device is at the time when the peripheral device is plugged into the USB and the enumeration process occurs. Thus, to alter the configuration or personality of a peripheral device, such as downloading new code or configuration information into the memory of the peripheral device, the host computer system must detect a peripheral device connection or a disconnection and then a reconnection.</p> <p style="padding-left: 40px;">The USB provides a number of advantages, as described above, over standard peripheral device connection techniques. The USB, however, does not provide a system and method for easily altering the configuration data for a peripheral device. In addition, the USB also does not provide a method for easily changing the software device driver associated with a particular peripheral device.</p> <p>Thus, there is a need for a system and method for interfacing to a universal serial bus which avoids these and other problems of known systems and methods, and it is to this end that the present invention is directed.</p> <p>APA, Ex. 1001, 1:66-2:19:</p> <p style="padding-left: 40px;">When a peripheral device is first connected to the USB and the host computer through a standard USB</p>

U.S. Patent 6,593,770	Correspondence to Admitted Prior Art (Ex. 1001) and Yap (Ex. 1002)
	<p>communications port, the presence of the connected peripheral device is detected and a configuration process of the USB for the connected peripheral device, known as device enumeration, begins. The enumeration process assigns a unique USB address to the connected peripheral device, queries the connected peripheral device about its requirements and capabilities, writes data about the connected peripheral device into the host computer's operating system, and loads the appropriate software device driver from a storage location into the host computer's operating system. During the query, a data table stored in the peripheral device, which contains the particular peripheral device's configuration information, is read from the peripheral device into the host computer's memory. Upon completion of the enumeration process, the connected peripheral device is recognized by the host computer's operating system and may be used by application software being executed by the microprocessor of the host computer. The association of the device with the software device driver cannot be subsequently changed.</p> <p>APA, Ex. 1001, 4:35-50:</p> <p>When the peripheral device is initially connected to the USB, an enumeration process is conducted in which the host computer determines the characteristics of the peripheral device by receiving the configuration information from the memory 38 within the peripheral device, and configures the USB according to the characteristics of the peripheral device. As shown, the configuration information about the characteristics of the peripheral device in a conventional USB system is stored in a non-volatile memory 38 on the peripheral device. The data about the characteristics of the peripheral device is programmed into the non-volatile memory at the factory, and the characteristics of the peripheral device may not be easily altered. In addition, the memory in the peripheral device stores all of the configuration information about the</p>

U.S. Patent 6,593,770	Correspondence to Admitted Prior Art (Ex. 1001) and Yap (Ex. 1002)
	peripheral device which may require a large amount of memory in the peripheral device.
[b] a first circuit configured to download information for a second configuration from the host computer into the peripheral device over the computer bus; and	<p style="text-align: center;"><b>APA</b></p> <p>APA, Ex. 1001, 2:20-39:</p> <p style="padding-left: 40px;">In a serial bus system, such as the USB, the only opportunity for associating software device drivers with a peripheral device is at the time when the peripheral device is plugged into the USB and the enumeration process occurs. Thus, to alter the configuration or personality of a peripheral device, such as downloading new code or configuration information into the memory of the peripheral device, the host computer system must detect a peripheral device connection or a disconnection and then a reconnection.</p> <p style="text-align: center;">. . .</p> <p>Thus, there is a need for a system and method for interfacing to a universal serial bus which avoids these and other problems of known systems and methods, and it is to this end that the present invention is directed.</p>
[c] a second circuit configured to electronically simulate a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device to said second configuration	<p style="text-align: center;"><b>APA</b></p> <p>APA, Ex. 1001, 2:20-39:</p> <p style="padding-left: 40px;">In a serial bus system, such as the USB, the only opportunity for associating software device drivers with a peripheral device is at the time when the peripheral device is plugged into the USB and the enumeration process occurs. Thus, to alter the configuration or personality of a peripheral device, such as downloading new code or configuration information into the memory of the peripheral device, the host computer system must detect a peripheral device connection or a disconnection and then a reconnection.</p> <p style="text-align: center;">. . .</p> <p>Thus, there is a need for a system and method for interfacing to a universal serial bus which avoids these and other problems of known systems and methods, and it is to this end that the present invention is directed.</p>



U.S. Patent 6,593,770	Correspondence to Admitted Prior Art (Ex. 1001) and Yap (Ex. 1002)
	<p>APA, Ex. 1001, 4:51-61, discussing Prior Art Fig. 1 (“Once the enumeration process has been completed, the CPU of the host computer may load an appropriate software device driver 33 for the peripheral device and the software applications being executed by that CPU of the host computer may communicate with the peripheral device using the USB. When the first peripheral device is disconnected and another peripheral device is connected to the USB, the enumeration process for the new peripheral device may be conducted and another software device driver may be loaded. The configuration of the peripheral device cannot be easily altered.”).</p> <p style="text-align: center;"><b>YAP</b></p> <p>Yap, Ex. 1002, 1:21-24 (“This invention relates to a method and apparatus for allowing a USB device to recover from a malfunction condition.”).</p> <p>Yap, Ex. 1002, 1:27-42:</p> <p style="padding-left: 40px;">USB is a peripheral bus standard that allows computer peripherals to be attached to a personal computer without the need for specialized cards or other vendor specific hardware attachments. . . . Information about the USB standard, including the USB specification v1.0, incorporated herein by reference, for building USB compliant devices, is currently available free of charge over the Internet.</p> <p>Yap, Ex. 1002, 1:43-67:</p> <p style="padding-left: 40px;">However, a malfunction condition may occur in a USB device, such as a plug-and-play device, wherein the USB device after being configured by the host computer may malfunction and stop communicating with the host computer due to problems, such as transmission errors, USB protocol errors, bugs in the host operating system or</p>

U.S. Patent 6,593,770	Correspondence to Admitted Prior Art (Ex. 1001) and Yap (Ex. 1002)
	<p>device firmware, etc. For example, a host operating system may terminate the function of the USB device, which may be busy at the moment or fails to acknowledge incoming data packets more than three times, for not communicating with the host computer. The above situation is referred to as a “brown out” condition.</p> <p>According to the USB specification v1 .0, page 201, the host operating system is supposed to record the last error type without trying to re-establish communications with the non-communicating USB device. When this occurs, (1) the user may have to re-boot the USB device or physically disconnect and then re-connect the USB device to allow the host computer to recognize and then re-configure the USB device . . . . The first method defeats the whole purpose of plug-and-play technology . . . .</p> <p>Yap, Ex. 1002, 2:7-24:</p> <p>Accordingly, one object of the present invention is to provide a method and apparatus for recovering from a USB device brown out condition which requires no user intervention.</p> <p>. . .</p> <p>It is also an object of the present invention to provide a method and apparatus for recovering from a USB device brown out condition without a need to re-boot the USB device or physically disconnect and then re-connect the USB device.</p> <p>Yap, Ex. 1002, Fig. 2:</p>

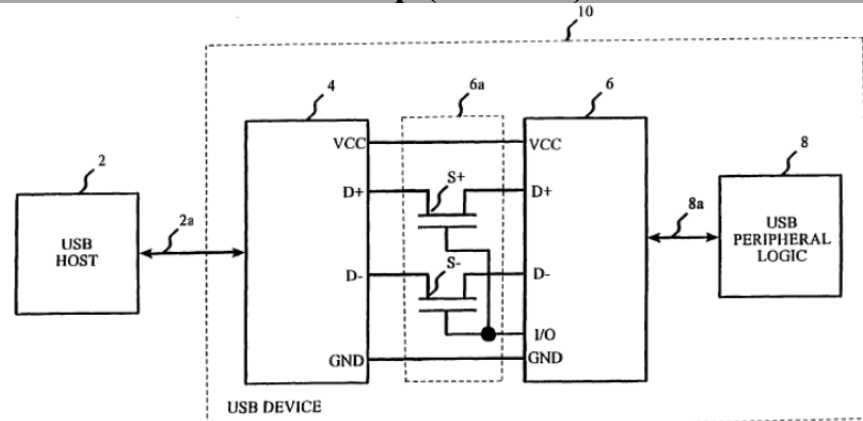


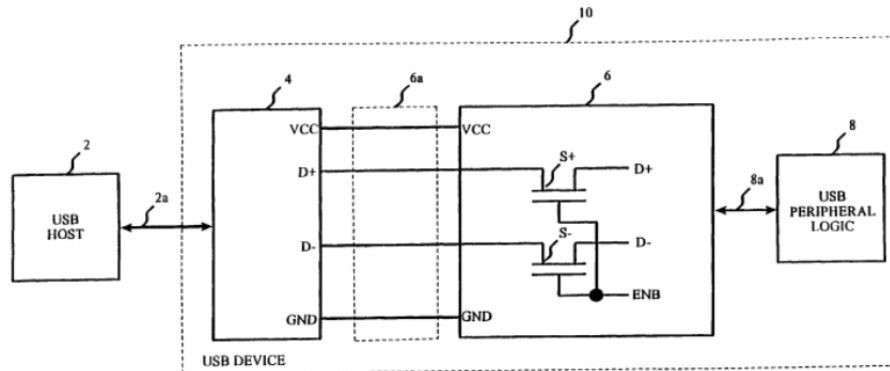
Fig. 2

Yap, Ex. 1002, 3:60-4:23:

In FIG. 2, a first embodiment of the USB device 10 of FIG. 1 further includes switching devices S+ and S-, such as transistors, contact switches, etc., coupled to positive data (D+) and negative data (D-) lines of the signal lines 6a. . . . [W]hen the USB micro-controller 6 drives the I/O pin to an appropriate logic state, the D+ and D- data lines may be opened or shorted via switching devices S+ and S-. By disconnecting the D+ and D- data lines via switching devices S+ and S-, a physical removal of the USB device 10 may be simulated in order to allow the USB host to re-configure the USB device 10 during a brown out condition. . . . Firmware in the USB micro-controller 6 keeps the data lines connected via switching devices S+ and S- during normal operation. However, when a brown out condition is detected, as will be described later, the USB micro-controller 6 opens the data lines via the switching devices S+ and S- for a duration greater than 2.5 micro-seconds and then reconnects them again. This procedure, for example, emulates the disconnect and re-connect procedure as specified in the USB specification v1.0, page 116.

Yap, Ex. 1002, Fig. 3:

<b>U.S. Patent 6,593,770</b>	<b>Correspondence to Admitted Prior Art (Ex. 1001) and Yap (Ex. 1002)</b>
----------------------------------	-------------------------------------------------------------------------------



*Fig. 3*

Yap, Ex. 1002, 4:24-38 (“FIG. 3, is a second embodiment of the USB device 10 wherein the switching devices S+ and S-, of FIG. 2 are included within the USB micro-controller 6. . . . Otherwise, the operation of the circuit of FIG. 3 is identical to the operation of the circuit of FIG. 2.”).

[d] while supplying electrical power to said peripheral device.

**YAP**

Yap, Ex.1002, Figs. 2-3

Yap, Ex. 1002, 3:64-67:

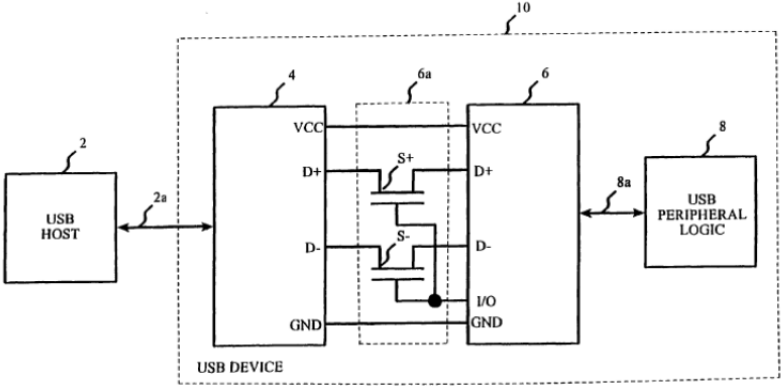
Please note that only one pair of complementary data lines of a plurality of complementary data lines and VCC and GND connections are shown in FIG. 2 for simplicity.

Yap, Ex. 1002, 4:10-16:

In addition, the general purpose I/O pin of the USB micro-controller 6 is configured such that during and after a reset condition due to power up the data lines stay connected (e.g., the I/O pin enables switching devices S+ and during and after reset). Firmware in the USB micro-controller 6 keeps the data lines connected via switching devices S+ and S- during normal operation.

Yap, Ex. 1002, 4:3-23:

<p><b>U.S. Patent 6,593,770</b></p>	<p><b>Correspondence to Admitted Prior Art (Ex. 1001) and Yap (Ex. 1002)</b></p>
	<p>Accordingly, when the USB micro-controller 6 drives the I/O pin to an appropriate logic state, the D+ and D- data lines may be opened or shorted via switching devices S+ and S-. By disconnecting the D+ and D- data lines via switching devices S+ and S-, a physical removal of the USB device 10 may be simulated in order to allow the USB host to re-configure the USB device 10 during a brown out condition.</p>
<p><b>Claim 5</b></p>	
<p>The system of claim 1, wherein said computer bus comprises a Universal Serial Bus.</p>	<p style="text-align: center;"><b>APA</b></p> <p>APA, Ex. 1001, 2:20-39:</p> <p style="padding-left: 40px;">In a serial bus system, such as the USB, the only opportunity for associating software device drivers with a peripheral device is at the time when the peripheral device is plugged into the USB and the enumeration process occurs. Thus, to alter the configuration or personality of a peripheral device, such as downloading new code or configuration information into the memory of the peripheral device, the host computer system must detect a peripheral device connection or a disconnection and then a reconnection.</p> <p style="text-align: center;">. . .</p> <p>Thus, there is a need for a system and method for interfacing to a universal serial bus which avoids these and other problems of known systems and methods, and it is to this end that the present invention is directed.</p>
<p><b>Claim 7</b></p>	
<p>The system of claim 5, wherein said information for said second configuration comprises (i) configuration data and (ii) an executable code.</p>	<p style="text-align: center;"><b>APA</b></p> <p>APA, Ex. 1001, 2:20-39:</p> <p style="padding-left: 40px;">In a serial bus system, such as the USB, the only opportunity for associating software device drivers with a peripheral device is at the time when the peripheral device is plugged into the USB and the enumeration process occurs. Thus, to alter the configuration or personality of a peripheral device, such as downloading new code or configuration information into the memory of the</p>

U.S. Patent 6,593,770	Correspondence to Admitted Prior Art (Ex. 1001) and Yap (Ex. 1002)
	<p>peripheral device, the host computer system must detect a peripheral device connection or a disconnection and then a reconnection.</p> <p>...</p> <p>Thus, there is a need for a system and method for interfacing to a universal serial bus which avoids these and other problems of known systems and methods, and it is to this end that the present invention is directed.</p>
Claim 10	
<p>The system of claim 1, wherein said second circuit comprises a reset circuit configured to reset the first or second configuration of the peripheral device.</p>	<p style="text-align: center;"><b>YAP</b></p> <p>Yap, Ex. 1002, Fig. 2:</p>  <p style="text-align: center;"><i>Fig. 2</i></p> <p>Yap, Ex. 1002, 3:60-4:23:</p> <p>In FIG. 2, a first embodiment of the USB device 10 of FIG. 1 further includes switching devices S+ and S-, such as transistors, contact switches, etc., coupled to positive data (D+) and negative data (D-) lines of the signal lines 6a. . . . [W]hen the USB micro-controller 6 drives the I/O pin to an appropriate logic state, the D+ and D- data lines may be opened or shorted via switching devices S+ and S-. By disconnecting the D+ and D- data lines via switching devices S+ and S-, a physical removal of the USB device 10 may be simulated in order to allow the USB host to</p>

U.S. Patent 6,593,770	Correspondence to Admitted Prior Art (Ex. 1001) and Yap (Ex. 1002)
	<p>re-configure the USB device 10 during a brown out condition. . . . Firmware in the USB micro controller 6 keeps the data lines connected via switching devices S+ and S- during normal operation. However, when a brown out condition is detected, as will be described later, the USB micro-controller 6 opens the data lines via the switching devices S+ and S- for a duration greater than 2.5 microseconds and then reconnects them again. This procedure, for example, emulates the disconnect and re-connect procedure as specified in the USB specification v1.0, page 116.</p> <p>Yap, Ex. 1002, Fig. 3:</p> <p style="text-align: center;"><i>Fig. 3</i></p> <p>Yap, Ex. 1002, 4:24-38 (“FIG. 3, is a second embodiment of the USB device 10 wherein the switching devices S+ and S-, of FIG. 2 are included within the USB micro-controller 6. . . . Otherwise, the operation of the circuit of FIG. 3 is identical to the operation of the circuit of FIG. 2.”).</p>
Claim 11	
[a] A method for reconfiguring a peripheral device having a first configuration connected by a computer bus to	See discussion above with respect to claim element 1[a].

U.S. Patent 6,593,770	Correspondence to Admitted Prior Art (Ex. 1001) and Yap (Ex. 1002)
a host computer, the method comprising the steps of:	
[b] (A) downloading second configuration from the host computer into the peripheral device over the computer bus; and	<i>See discussion above with respect to claim element 1[b].</i>
[c] (B) electronically simulating a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device to said second configuration	<i>See discussion above with respect to claim element 1[c].</i>
[d] while supplying electrical power to said peripheral device.	<i>See discussion above with respect to claim element 1[d].</i>
<b>Claim 15</b>	
The method of claim 11, wherein step (A)	<i>See discussion above with respect to claim 5.</i>



U.S. Patent 6,593,770	Correspondence to Admitted Prior Art (Ex. 1001) and Yap (Ex. 1002)
comprises communicating said information for the second configuration to the peripheral device over a Universal Serial Bus.	
Claim 16	
The method of claim 11, wherein said information for the second configuration comprises (i) configuration data and (ii) an executable code.	<i>See</i> discussion above with respect to claim 7.
Claim 17	
The method of claim 11, wherein step (B) comprises electronically resetting the configuration of the peripheral device, controllable by the peripheral device.	<i>See</i> discussion above with respect to claim 10.

**B. Ground 2: Claims 2, 3, 12, and 13 are unpatentable under 35 U.S.C. § 103, as being obvious over the Admitted Prior Art (APA), U.S. Patent No. 6,073,193 to Yap, and U.S. Patent No. 5,628,028 to Michelson**

	<b>Correspondence to APA (Ex. 1001), Yap (Ex. 1002), and Michelson (Ex. 1003)</b>
<b>Claim 2</b>	
<p>The system of claim 1, wherein said first configuration is a generic configuration assigned to the peripheral device and said second configuration comprises any one of a plurality of unique manufacturer configurations.</p>	<p style="text-align: center;"><b>MICHELSON</b></p> <p>Michelson, Ex. 1003, 2:34-4:8:</p> <p>Referring also to FIG. 2, when PCMCIA card 14 card connector 28 is inserted in PCMCIA host socket 18 of host computer 12, PCMCIA adapter 16 recognizes (step 44) the insertion and interrupts (step 46) processor 22. Alternatively, if PCMCIA card 14 is inserted while host computer 12 is turned off (i.e., powered-down), host computer 12 learns of the existence of PCMCIA card 14 during the power-on procedure. Processor 22 then executes (step 48) Card and Socket Services 38 (C&amp;SS) software resident in host memory 24 and through PCMCIA interface chip 26 reads CIS data from CIS EEPROM 30. As a minimum, the CIS data must sufficiently identify the PCMCIA card to the host, to enable the processor 22 to configure the host computer 12 and the PCMCIA card 14 to operate together and to enable the processor to select the appropriate application software 40 from host memory 24. The CIS data specifically identify the card manufacturer (e.g., Data Translation, Inc.) and card identification (ID) number and includes a variety of set-up information, including base address, interrupt level, size of address window, and other information regarding the card's functionality, as specified by release 2.1. The CIS data are entered into the EEPROM at the time of card manufacture and are not thereafter changed. Hence, configuration of host computer 12 and PCMCIA card 14 is completed without the use of card controller 32. Processor 22 then executes (step 50) the application software 40 resident in host memory 24 that corresponds to PCMCIA card 14. The</p>

	<p align="center"><b>Correspondence to APA (Ex. 1001), Yap (Ex. 1002), and Michelson (Ex. 1003)</b></p>
	<p>application software 40 causes the processor to either select a default FPGA programming data file 42 from host memory 24 that corresponds to a particular application for PCMCIA card 14 or request input from the user as to which FPGA programming data file 42 is to be selected from host memory 24. Processor 22 then sends (step 52) the data from the selected FPGA programming data file 42 through PCMCIA adapter 16 to PCMCIA interface chip 26. Interface chip 26 then programs (step 54) a field programmable gate array (FPGA, not shown in FIG. 1) within card controller 32 by loading the data from the FPGA programming data file 42 into the FPGA. Where the application software causes the processor to select a default data programming file, PCMCIA card 14 and host computer 12 are made operable (step 56) without user intervention.</p> <p>Michelson, Ex. 1003, 6:61-65:</p> <p>The FPGA programming data files can be supplied with the PCMCIA card or new, additional, or updated FPGA programming data files can be obtained at a later time. Similarly, users can create their own FPGA programming data files or make modifications as desired.</p>
<p align="center"><b>Claim 3</b></p>	
<p>The system of claim 2, wherein said first circuit is configured to (i) read an identification code from the peripheral device and (ii) select said second configuration based on said identification code.</p>	<p align="center"><b>MICHELSON</b></p> <p>Michelson, Ex. 1003, 2:34-4:8:</p> <p>Referring also to FIG. 2, when PCMCIA card 14 card connector 28 is inserted in PCMCIA host socket 18 of host computer 12, PCMCIA adapter 16 recognizes (step 44) the insertion and interrupts (step 46) processor 22. Alternatively, if PCMCIA card 14 is inserted while host computer 12 is turned off (i.e., powered-down), host computer 12 learns of the existence of PCMCIA card 14 during the power-on procedure. Processor 22 then executes (step 48) Card and Socket Services 38 (C&amp;SS)</p>

	<p align="center"><b>Correspondence to APA (Ex. 1001), Yap (Ex. 1002), and Michelson (Ex. 1003)</b></p>
	<p>software resident in host memory 24 and through PCMCIA interface chip 26 reads CIS data from CIS EEPROM 30. As a minimum, the CIS data must sufficiently identify the PCMCIA card to the host, to enable the processor 22 to configure the host computer 12 and the PCMCIA card 14 to operate together and to enable the processor to select the appropriate application software 40 from host memory 24. The CIS data specifically identify the card manufacturer (e.g., Data Translation, Inc.) and card identification (ID) number and includes a variety of set-up information, including base address, interrupt level, size of address window, and other information regarding the card's functionality, as specified by release 2.1. The CIS data are entered into the EEPROM at the time of card manufacture and are not thereafter changed. Hence, configuration of host computer 12 and PCMCIA card 14 is completed without the use of card controller 32.</p> <p>Processor 22 then executes (step 50) the application software 40 resident in host memory 24 that corresponds to PCMCIA card 14. The application software 40 causes the processor to either select a default FPGA programming data file 42 from host memory 24 that corresponds to a particular application for PCMCIA card 14 or request input from the user as to which FPGA programming data file 42 is to be selected from host memory 24. Processor 22 then sends (step 52) the data from the selected FPGA programming data file 42 through PCMCIA adapter 16 to PCMCIA interface chip 26. Interface chip 26 then programs (step 54) a field programmable gate array (FPGA, not shown in FIG. 1) within card controller 32 by loading the data from the FPGA programming data file 42 into the FPGA. Where the application software causes the processor to select a default data programming file, PCMCIA card 14 and host computer 12 are made operable (step 56) without user intervention.</p>

	<b>Correspondence to APA (Ex. 1001), Yap (Ex. 1002), and Michelson (Ex. 1003)</b>
	<p>Michelson, Ex. 1003, 6:61-65:</p> <p>The FPGA programming data files can be supplied with the PCMCIA card or new, additional, or updated FPGA programming data files can be obtained at a later time. Similarly, users can create their own FPGA programming data files or make modifications as desired.</p>
<b>Claim 12</b>	
The method of claim 11, wherein said first configuration comprises a generic configuration assigned to the peripheral device and said second configuration comprises any one of a plurality of unique manufacturer configurations.	<i>See discussion above with respect to claim 2.</i>
<b>Claim 13</b>	
The method of claim 11, wherein step (A) comprises: reading an identification code from the peripheral device, and selecting said second configuration based on said identification code.	<i>See discussion above with respect to claim 3.</i>



**C. Ground 3: Claims 1-3, 10, 11, 12, 13, 16, 17, 18, and 20 are unpatentable under 35 U.S.C. § 103 as being obvious over Michelson (U.S. Patent No. 5,628,028), PCCextend 100 User’s Manual (“PCCextend”), and Davis (U.S. Patent No. 5,862,393)**

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
<b>Claim 1</b>	
<p>[a] A system for reconfiguring a peripheral device having a first configuration connected by a computer bus to a host computer, the system comprising:</p>	<p style="text-align: center;"><b>MICHELSON</b></p> <p>Michelson, Ex. 1003, 1:7-16:</p> <p style="padding-left: 40px;">This invention relates to programming and reprogramming the hardware configuration of a (PCMCIA) card. Personal computer memory card international association (PCMCIA) cards are computer cards that meet the minimum compliance requirements of the PCMCIA standard (e.g., release 2.1, which is hereby incorporated by reference). PCMCIA cards are typically used to add functionality or memory to a personal, portable, or desktop computer (i.e., host computer).</p> <p>Michelson, Ex. 1003, 1:28-36:</p> <p style="padding-left: 40px;">A typical PCMCIA card includes a standard PCMCIA connector connected to a PCMCIA interface circuit through a standard PCMCIA bus. The PCMCIA interface circuit operates according to the standard PCMCIA protocol to send data to and receive data from a host computer. The typical PCMCIA card also includes a PCMCIA card controller that sends data to and receives data from the PCMCIA interface circuit and controls the operation of the functional hardware on the card.”).</p> <p>Michelson, Ex. 1003, 2:3-16:</p> <p style="padding-left: 40px;">In general, the invention includes a PCMCIA card having an FPGA based card controller that is programmed with FPGA programming data stored on a host computer through a standard PCMCIA bus. Storing FPGA programming data on the host computer allows a user access to a practically unlimited number of FPGA programming data files to program</p>

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>and reprogram the FPGA of the PCMCIA FPGA based card controller for different applications and permits a user to supplement, update, improve, or otherwise modify operation for existing applications. Additionally, storing the FPGA programming data files on the host computer saves valuable PCMCIA card real estate, reduces the amount of power required by the card during FPGA programming, and reduces the cost of the PCMCIA hardware.”).</p> <p>Michelson, Ex. 1003, 2:17-42:</p> <p>In preferred embodiments, the host computer includes a PCMCIA adapter circuit coupled to a PCMCIA host socket which is mechanically and electrically connected to a PCMCIA card connector on the PCMCIA card. A PCMCIA interface circuit is connected to the PCMCIA card connector on the PCMCIA card. Using Card and Socket Services software stored in host memory, the host processor reads Card Information Structure (CIS) data from a memory device, such as an EEPROM, on the PCMCIA card and configures the host computer and PCMCIA card to operate together. Additionally, using application software stored in host memory, the processor selects an FPGA programming data file from host memory and sends data from the selected FPGA programming data file through the PCMCIA adapter circuit to the PCMCIA interface circuit. The PCMCIA interface circuit loads the data into a PCMCIA card controller FPGA to program the FPGA. When an error or a different user application is detected or when a user creates a new FPGA programming data file or modifies an existing FPGA programming data file, the processor is instructed to select another FPGA programming data file from host memory. The processor then sends data from the newly selected FPGA programming data file to the PCMCIA interface circuit, and the PCMCIA interface circuit loads the data into the PCMCIA card controller FPGA to reprogram the FPGA.</p> <p>Michelson, Ex. 1003, Fig. 1:</p>



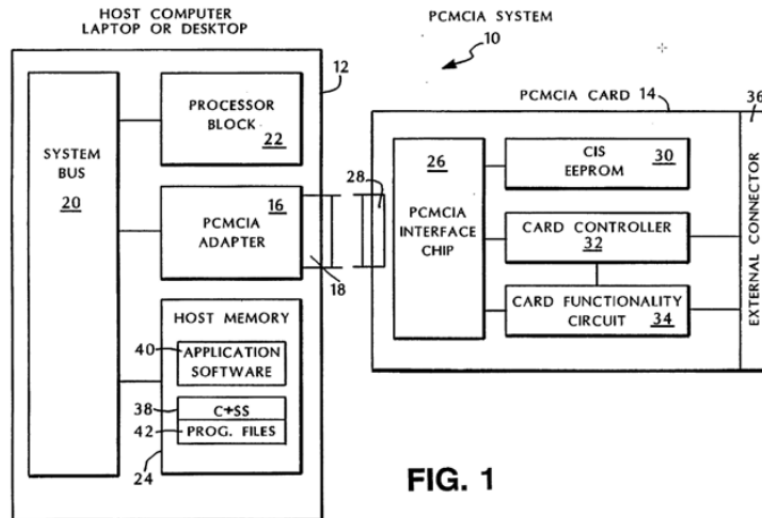


FIG. 1

Michelson, Ex. 1003, 2:62-3:8:

Referring to FIG. 1, a PCMCIA system 10 includes a host computer 12 and a PCMCIA card 14. Within host computer 12, a PCMCIA adapter 16, connected to a standard 68 pin PCMCIA host socket 18 . . . , is coupled to a system bus 20 that interconnects PCMCIA adapter 16, a host processor 22, and a host memory 24. Within PCMCIA card 14, a PCMCIA interface chip 26 . . . , connected to a standard 68 pin PCMCIA card connector 28 . . . , is coupled to a Card Information Structure (CIS) EEPROM 30 . . . , a PCMCIA card controller 32, and a card functionality circuit 34.

Michelson, Ex. 1003, Fig. 3:

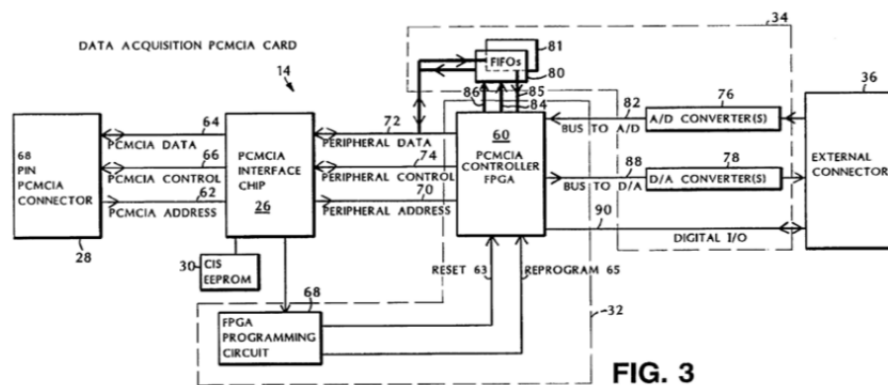
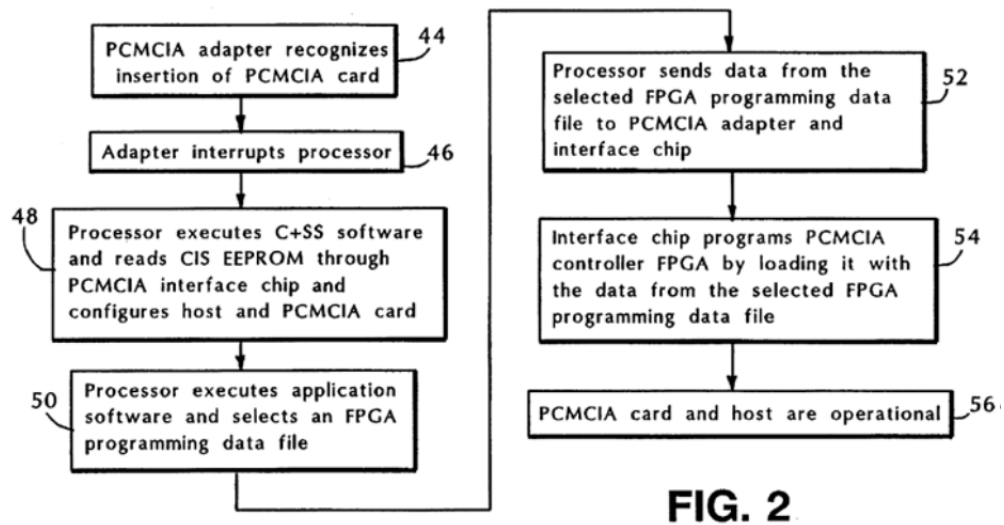


FIG. 3

Michelson, Ex. 1003, 4:9-22

Referring to FIG. 3, card controller 32 includes a PCMCIA card controller FPGA 60 (e.g., part number XC3042TQ 100-100, manufactured by Xilinx, as described in Xilinx Programmable Logic Data Book, which is hereby incorporated by reference). Through a standard PCMCIA bus (i.e., PCMCIA address lines 62, data lines 64, and control lines 66) connected to PCMCIA connector 28, interface chip 26 receives FPGA programming data from host computer 12 (FIG. 1). Interface chip 26 initiates FPGA 60 programming through FPGA programming circuit 68, which drives reset line 63 and reprogram line 65, and completes FPGA 60 programming by loading the FPGA programming data into FPGA 60 through peripheral data lines 72.

Michelson, Ex. 1003, Fig. 2:



Michelson, Ex. 1003, 2:34-58:

Referring also to FIG. 2, when PCMCIA card 14 card connector 28 is inserted in PCMCIA host socket 18 of host computer 12, PCMCIA adapter 16 recognizes (step 44) the

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>insertion and interrupts (step 46) processor 22. Alternatively, if PCMCIA card 14 is inserted while host computer 12 is turned off (i.e., powered-down), host computer 12 learns of the existence of PCMCIA card 14 during the power-on procedure. Processor 22 then executes (step 48) Card and Socket Services 38 (C&amp;SS) software resident in host memory 24 and through PCMCIA interface chip 26 reads CIS data from CIS EEPROM 30. As a minimum, the CIS data must sufficiently identify the PCMCIA card to the host, to enable the processor 22 to configure the host computer 12 and the PCMCIA card 14 to operate together and to enable the processor to select the appropriate application software 40 from host memory 24. The CIS data specifically identify the card manufacturer (e.g., Data Translation, Inc.) and card identification (ID) number and includes a variety of set-up information, including base address, interrupt level, size of address window, and other information regarding the card's functionality, as specified by release 2.1. The CIS data are entered into the EEPROM at the time of card manufacture and are not thereafter changed. Hence, configuration of host computer 12 and PCMCIA card 14 is completed without the use of card controller 32.</p>
<p>[b] a first circuit configured to download information for a second configuration from the host computer into the peripheral device over the computer bus; and</p>	<p style="text-align: center;"><b>MICHELSON</b></p> <p>Michelson, Ex. 1003, 3:32-40:</p> <p style="padding-left: 40px;">The typical PCMCIA card also includes a PCMCIA card controller that sends data to and receives data from the PCMCIA interface circuit and controls the operation of the functional hardware on the card. For example, if the PCMCIA card is a memory card, then the functional hardware is memory (e.g., a bank of random access memory (RAM) chips (static or dynamic) or a hard disk drive) and the PCMCIA card controller controls reading and writing to the memory.</p> <p>Michelson, Ex. 1003, 3:59-4:22:</p> <p style="padding-left: 40px;">Processor 22 then executes (step 50) the application software 40 resident in host memory 24 that corresponds to PCMCIA card 14. The application software 40 causes the</p>

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>processor to either select a default FPGA programming data file 42 from host memory 24 that corresponds to a particular application for PCMCIA card 14 or request input from the user as to which FPGA programming data file 42 is to be selected from host memory 24. Processor 22 then sends (step 52) the data from the selected FPGA programming data file 42 through PCMCIA adapter 16 to PCMCIA interface chip 26. Interface chip 26 then programs (step 54) a field programmable gate array (FPGA, not shown in FIG. 1) within card controller 32 by loading the data from the FPGA programming data file 42 into the FPGA. Where the application software causes the processor to select a default data programming file, PCMCIA card 14 and host computer 12 are made operable (step 56) without user intervention.</p> <p>Referring to FIG. 3, card controller 32 includes a PCMCIA card controller FPGA 60 (e.g., part number XC3042TQ 100-100, manufactured by Xilinx, as described in Xilinx Programmable Logic Data Book, which is hereby incorporated by reference). Through a standard PCMCIA bus (i.e., PCMCIA address lines 62, data lines 64, and control lines 66) connected to PCMCIA connector 28, interface chip 26 receives FPGA programming data from host computer 12 (FIG. 1). Interface chip 26 initiates FPGA 60 programming through FPGA programming circuit 68, which drives reset line 63 and reprogram line 65, and completes FPGA 60 programming by loading the FPGA programming data into FPGA 60 through peripheral data lines 72.</p> <p>Michelson, Ex. 1003, 5:37-5 1:</p> <p>For a variety of reasons, including the detection of a PCMCIA card controller 60 malfunction (e.g., error condition detected) or the detection of a user request to change the PCMCIA card application (e.g., software interrupt), PCMCIA card controller 60 may be reprogrammed. The host computer executes application software 40 (FIG. 1) in host memory 24 to select a new FPGA programming data file 42 from host memory 24 and then sends the data from the newly selected</p>

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>FPGA programming data file 42 through PCMCIA adapter 16 to PCMCIA interface chip 26. Interface chip 26 uses FPGA programming circuit 68 to reset PCMCIA card controller FPGA 60 and enable reprogramming, and interface chip 26 completes reprogramming by loading the data from the newly selected FPGA programming data file into card controller FPGA 60.</p>
<p>[c] a second circuit configured to electronically simulate a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device to said second configuration</p>	<p style="text-align: center;"><b>MICHELSON</b></p> <p>Michelson, Ex. 1003, 4:9-22:</p> <p>Referring to FIG. 3, card controller 32 includes a PCMCIA card controller FPGA 60 (e.g., part number XC3042TQ100-100, manufactured by Xilinx, as described in Xilinx Programmable Logic Data Book, which is hereby incorporated by reference). Through a standard PCMCIA bus (i.e., PCMCIA address lines 62, data lines 64, and control lines 66) connected to PCMCIA connector 28, interface chip 26 receives FPGA programming data from host computer 12 (FIG. 1). Interface chip 26 initiates FPGA 60 programming through FPGA programming circuit 68, which drives reset line 63 and reprogram line 65, and completes FPGA 60 programming by loading the FPGA programming data into FPGA 60 through peripheral data lines 72.</p> <p>Michelson, Ex. 1003, 5:37-5 1:</p> <p>For a variety of reasons, including the detection of a PCMCIA card controller 60 malfunction (e.g., error condition detected) or the detection of a user request to change the PCMCIA card application (e.g., software interrupt), PCMCIA card controller 60 may be reprogrammed. The host computer executes application software 40 (FIG. 1) in host memory 24 to select a new FPGA programming data file 42 from host memory 24 and then sends the data from the newly selected FPGA programming data file 42 through PCMCIA adapter 16 to PCMCIA interface chip 26. Interface chip 26 uses FPGA programming circuit 68 to reset PCMCIA card controller FPGA 60 and enable reprogramming, and interface chip 26</p>

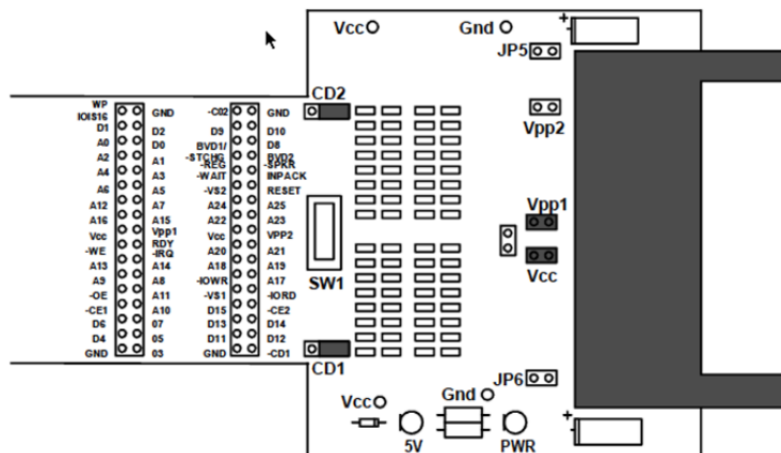
U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>completes reprogramming by loading the data from the newly selected FPGA programming data file into card controller FPGA 60.</p> <p>Michelson, Ex. 1003, 5:52-6:60:</p> <p>For example, instead of simply storing all the data received from the sensors in the FIFO before sending it to the host computer, card controller FPGA 60 can be reprogrammed with additional functionality that formats the data received from the sensors according to the host computer's requirements before storing the data in the FIFO. Similarly, card controller FPGA 60 can be reprogrammed with additional functionality that analyzes the sensor data and interrupts the host computer when predetermined thresholds are exceeded. In such a system, the host computer need not interact with the PCMCIA card unless a predetermined threshold is exceeded.</p> <p>For example, a data acquisition PCMCIA card may be coupled to a temperature controller and sensors for determining the temperature of a room. Such a PCMCIA card continually receives data from the sensors and, in a simple data acquisition card, the host computer periodically reads the data acquisition FIFO and analyzes the data to determine if predetermined temperature thresholds have been exceeded. The PCMCIA card could be reprogrammed to analyze the data received from the sensors and interrupt the host computer when a predetermined temperature threshold is exceeded. Hence, the host computer would only read the PCMCIA card FIFO when notified that a threshold had been exceeded.</p> <p>Moreover, the operation of the PCMCIA card controller may need to change for different user applications. For example, if a temperature controller is moved to a smaller room where temperature fluctuates more quickly, the PCMCIA card controller needs to be reprogrammed to accept data from the temperature sensors more frequently. In such a situation, the user notifies the processor of a change in application and the processor selects a corresponding FPGA programming data file and sends the data to the PCMCIA interface which reprograms</p>

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>the PCMCIA card controller FPGA by loading the FPGA with the new FPGA programming data.</p> <p>Typically host memory is very large and is supplemented with extended memory (not shown). Hence, a practically unlimited number of FPGA programming data files can be stored within the host computer and made available to the user. The programming and reprogramming of the PCMCIA card controller FPGA is limited only by the size (i.e., capability) of the FPGA. The functionality of the PCMCIA card is limited only by the fixed hardware (i.e., the functional hardware) on the PCMCIA card (e.g., A/D or D/A converters, contacts on and configuration of the external connector, and FIFO size).</p> <p>For example, a PCMCIA card controller FPGA can be programmed to function as an I/O card controller, a data acquisition card controller, a fax/modem card controller, or a memory card controller; however, the PCMCIA card can only function as these card types if the additional functional hardware is available on the card. For an I/O card controller, the FPGA can be programmed (and reprogrammed) with the functionality required to transfer data between the host computer and the I/O bus (e.g., Small Computer System Interface (SCSI)), and, thus, the only additional functional hardware required is an external I/O bus connector and electrical conductors from the I/O bus connector to the FPGA. For a memory card, additional functional hardware typically includes a bank of static or dynamic RAM chips, ROM chips, flash memory, or a hard disk drive, and the addressing and refreshing functionality can be located within the PCMCIA card controller FPGA and, hence, reprogrammable. As an example, where a portion of a disk drive or a portion of a bank of RAMs becomes damaged and non-functional, the addressing functionality in the FPGA can be reprogrammed to address only the working portion of the memory hardware. For a fax/modem, additional functional hardware generally includes a phone connection, A/D and D/A converters, buffers, and amplifiers, and functionality controlling hardware, for instance, hardware controlling the baud rate, can be located within the FPGA and, thus, reprogrammable.</p>

**PCCextend**

PCCextend, Ex. 1004, p. 1 (“Using the PCCextend is relatively straightforward. The extender card is inserted into the desired slot in the host system. Then the PC Card under test is inserted into the card connector.”).

PCCextend, Ex. 1004, p. 1 (“Caution: Insertion and removal of the extender and PC card should be done with care. The PC Card’s fragile connectors may be broken or bent if improper force is used. Both card and extender should be inserted straight without any lateral movement or force.” (Italics and bold omitted.)).



PCCextend, Ex. 1004, pp. 3-4:

2.3 Using the PCCswitch

PCCextend 100 includes the PCCswitch, which can be used to momentarily interrupt the CD1 and CD2 signals. The PCCswitch is centrally located on the PCCextend 100 between the termination area and test points. When properly configured, the PCCswitch can interrupt the card detect signals (-CD 1 and -CD2) to simulate a card removal/insertion cycle. . . . When a card is inserted, CD1 and CD2 may be momentarily interrupted by pressing the PCCswitch.



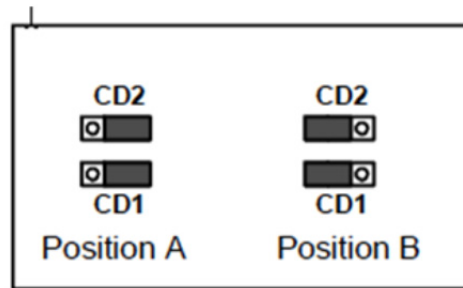


Figure 2.3-1 Card Detect Switch

To test the operation of the PCCswitch, be sure that your PC Card Software drivers are loaded. Momentarily press the PCCtest switch. Most software drivers will issue a removal beep followed by an insertion beep. The software may also remove power from the socket when the card is removed.

**DAVIS**

Davis, Ex. 1005, 1:20-33 (“A power management system typically operates to conserve electrical power consumption by reducing power requirements in response to a detected lack of activity by a computer or its devices.”).

Davis, Ex. 1005, 1:43-49:

A power management event typically comprises either a power-down or power-up event. A sequence of power-down and power-up events can cause a computer device to enter a default state or a random state based on the loss of configuration information. It is often necessary to supply configuration information to a device via its device driver in response to a sequence of power-down and power-up events.

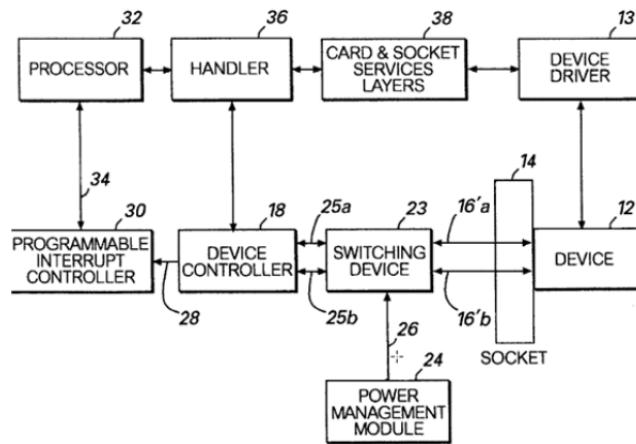
Davis, Ex. 1005, 2:21-32:

These removable devices can lose device configuration information in response to a power-down/power-up sequence in the absence of an appropriate power management system. Indeed, if the power management event is not communicated to

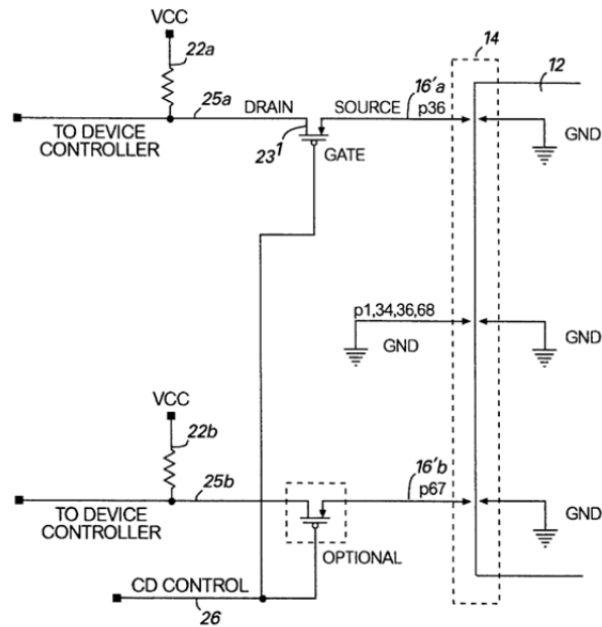
U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>the device driver, the only way to return a device that has lost its device configuration to a useful state is to restart or re-boot the computer system. The present invention solves these issues by using the device removal and insertion signals normally generated by the removal or insertion of a device to support a power management application and to advise a driver for a device about a power management event.</p> <p>Davis, Ex. 1005, 5:13-26:</p> <p>To achieve the desired power management function, it is often necessary to cause a removable device to enter a reduced power state when a device is inactive or placed in the idle state. However, a device will often lose its device configuration information in response to powering down that device. To fully use that device once power is restored, it is necessary to reconfigure the device with device configuration information. Specifically, it is desirable to communicate configuration information to a device via its device driver in response to restoring power. The present invention provides a solution to this problem by notifying the appropriate device driver of a power management event and by supplying device configuration information to a corresponding device in response to the restoration of power.</p> <p>Davis, Ex. 1005, 6:37-54:</p> <p>The state of the card detect lines 16a and 16b can be used by the device controller 18 to determine if a device 12 is connected to the socket 14. The device controller 18 determines that a device 12 is connected to the socket 14 when both card detect lines 16a and 16b transition from the logical high level to the logical low level. In contrast, in the event that one of the card detect lines 16a and 16b transitions from the logical low level to the logical high level, then the device controller 18 determines that a device 12 has been removed from the socket 14. In summary, a device insertion event is defined by a particular state for both of the card detect lines 16a and 16b,</p>

preferably each card detect line carrying a logical low level signal. A device removal event, however, is preferably defined by one of the card detect lines 16a and 16b transitioning to the logical high level. Those skilled in the art will appreciate that the removable device interface system described above is compatible with the standard specification for PCMCIA or PC Card devices.

Davis, Ex. 1005, Figs. 2 and 3:



**FIG. 2**



**FIG. 3**

Davis, Ex. 1005, 7:8-36:

FIG. 2 is a block diagram illustrating the components for an embodiment of the present invention. Referring now to FIG. 2, a switching device 23 is connected between the device 12 (via the socket 14) and the device controller 18. Specifically, the switching device 23 is connected to the socket 14 (and, if inserted, the device 12) by the card detect lines 16a' and 16b'. In addition, the switching device 23 is connected to the device controller 18 via the system advisory lines 25a and 25b. A power management module 24, which supports the power management function by controlling the power states of the computer and its devices, communicates with the switching device 23 via a control line 26. The control line 26 carries control signals output by the power management module 24 for controlling the operating state of the switching device 23.

The switching device 23 can operate in the open state in response to a particular control signal from the power management module 24, thereby breaking a signal path between the card detect lines 16a' and 16b' and system

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>advisory lines 25a and 25b. In the alternative, the switching device 23 can operate in the closed state in response to another control signal to maintain a signal connection between the card detect lines 16a' and 16b' and the system advisory line 25a and 25b. It will be appreciated that the switching device 23 can be implemented by an electronic switch, typically a field effect transistor (FET) or a bipolar transistor. Other types of electronic switches, however, can be used to implement the switching device 23, as is known in the art.</p> <p>Davis, Ex. 1005, 10:21-35:</p> <p>FIG. 3 is a schematic diagram for the preferred embodiment of the present invention. Turning now to FIGS. 2 and 3, the switching device 23 is preferably implemented by a FET device 23' connected between either (1) the combination of the card detect line 1 6a and the system advisory line of 25a or (2) the combination of the card detect line 1 6b and the system advisory line 25b. The FET 23' serves to manipulate a signal control path between the socket 14 and the device controller 18, and a device removal event can be represented by deactivating the FET and opening this single path. Likewise, a device insertion event can be represented by activating the FET and closing this signal path (while the device 12 is properly installed within the socket 14 and a card detect line extends between the socket and the device controller).</p> <p>Davis, Ex. 1005, 8:31-34:</p> <p>By manipulating the operating state of the switching device 23, information regarding device insertion or removal events can be communicated to the card controller 18. 8:31-34.</p> <p>Davis, Ex. 1005, 9:41-52:</p> <p>In response to a power-down event, the power management module outputs a particular control signal to the switching device 24, thereby causing the switching device to enter the open state or</p>

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>position. This interrupts the passage of signals from the card detect lines 16a' and 16b' to the device controller 18, thereby tricking the device controller to take actions in response to the apparent removal of the device 12. Significantly, the device 12 remains inserted within the socket 14, thereby leading to the presence of logical lower levels signals on the card detect lines 1 6a' and 1 6b' that represent a device insertion event.</p>
<p>[d] while supplying electrical power to said peripheral device.</p>	<p style="text-align: center;"><b>MICHELSON</b></p> <p>Michelson, Ex. 1003, 4:17-22; Interlace chip 26 initiates FPGA 60 programming through FPGA programming circuit 68, which drives reset line 63 and reprogram line 65, and completes FPGA 60 programming by loading the FPGA programming data into FPGA 60 through peripheral data lines 72.</p> <p>Michelson, Ex. 1003, 5:45-51 Interface chip 26 uses FPGA programming circuit 68 to reset PCMCIA card controller FPGA 60 and enable reprogramming, and interface chip 26 completes reprogramming by loading the data from the newly selected FPGA programming data file into card controller FPGA 60.</p> <p>Michelson, Ex. 1003, Figs. 1 and 3.</p> <p style="text-align: center;"><b>PCCEXTEND</b></p> <p>PCCextend, Ex. 1004, p. 3 (“The power LEDs are designed to indicate the presence of power on the Vcc supply pins”).</p> <p>PCCextend, Ex. 1004, p. 5 (“The software <u>may</u> also remove power from the socket when the card is removed”) (emphasis added).</p> <p>PCCextend, Ex. 1004, pp. 15-16.</p>
<b>Claim 2</b>	
<p>The system of claim 1, wherein said first</p>	<p style="text-align: center;"><b>MICHELSON</b></p> <p>Michelson, Ex. 1003, 2:34-4:8:  Referring also to FIG. 2, when PCMCIA card 14 card</p>

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
<p>configuration is a generic configuration assigned to the peripheral device and said second configuration comprises any one of a plurality of unique manufacturer configurations.</p>	<p>connector 28 is inserted in PCMCIA host socket 18 of host computer 12, PCMCIA adapter 16 recognizes (step 44) the insertion and interrupts (step 46) processor 22. Alternatively, if PCMCIA card 14 is inserted while host computer 12 is turned off (i.e., powered-down), host computer 12 learns of the existence of PCMCIA card 14 during the power-on procedure. Processor 22 then executes (step 48) Card and Socket Services 38 (C&amp;SS) software resident in host memory 24 and through PCMCIA interface chip 26 reads CIS data from CIS EEPROM 30. As a minimum, the CIS data must sufficiently identify the PCMCIA card to the host, to enable the processor 22 to configure the host computer 12 and the PCMCIA card 14 to operate together and to enable the processor to select the appropriate application software 40 from host memory 24. The CIS data specifically identify the card manufacturer (e.g., Data Translation, Inc.) and card identification (ID) number and includes a variety of set-up information, including base address, interrupt level, size of address window, and other information regarding the card's functionality, as specified by release 2.1. The CIS data are entered into the EEPROM at the time of card manufacture and are not thereafter changed. Hence, configuration of host computer 12 and PCMCIA card 14 is completed without the use of card controller 32.</p> <p>Processor 22 then executes (step 50) the application software 40 resident in host memory 24 that corresponds to PCMCIA card 14. The application software 40 causes the processor to either select a default FPGA programming data file 42 from host memory 24 that corresponds to a particular application for PCMCIA card 14 or request input from the user as to which FPGA programming data file 42 is to be selected from host memory 24. Processor 22 then sends (step 52) the data from the selected FPGA programming data file 42 through PCMCIA adapter 16 to PCMCIA interface chip 26. Interface chip 26 then programs (step 54) a field programmable gate array (FPGA, not shown in FIG. 1) within card controller 32 by loading the data from the FPGA programming data file 42 into the FPGA. Where the application software causes the processor to select a default data programming file, PCMCIA card 14 and</p>

<p><b>U.S. Patent 6,593,770</b></p>	<p><b>Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)</b></p>
	<p>host computer 12 are made operable (step 56) without user intervention.</p> <p>Michelson, Ex. 1003, 6:61-65:</p> <p>The FPGA programming data files can be supplied with the PCMCIA card or new, additional, or updated FPGA programming data files can be obtained at a later time. Similarly, users can create their own FPGA programming data files or make modifications as desired.</p>
<p><b>Claim 3</b></p>	
<p>The system of claim 2, wherein said first circuit is configured to (i) read an identification code from the</p>	<p style="text-align: center;"><b>MICHELSON</b></p> <p>Michelson, Ex. 1003, 2:34-4:8:</p> <p>Referring also to FIG. 2, when PCMCIA card 14 card connector 28 is inserted in PCMCIA host socket 18 of host computer 12, PCMCIA adapter 16 recognizes (step 44) the insertion and interrupts (step 46) processor 22. Alternatively, if PCMCIA card 14 is inserted while host computer 12 is turned off (i.e., powered-down), host computer 12 learns of the existence of PCMCIA card 14 during the power-on procedure. Processor 22 then executes (step 48) Card and Socket Services 38 (C&amp;SS) software resident in host memory 24 and through PCMCIA interface chip 26 reads CIS data from CIS EEPROM 30. As a minimum, the CIS data must sufficiently identify the PCMCIA card to the host, to enable the processor 22 to configure the host computer 12 and the PCMCIA card 14 to operate together and to enable the processor to select the appropriate application software 40 from host memory 24. The CIS data specifically identify the card manufacturer (e.g., Data Translation, Inc.) and card identification (ID) number and includes a variety of set-up information, including base address, interrupt level, size of address window, and other information regarding the card's functionality, as specified by release 2.1. The CIS data are entered into the EEPROM at the time of card manufacture and are not thereafter changed. Hence, configuration of host computer 12 and PCMCIA card 14 is completed without the use of card controller 32.</p>



<p><b>U.S. Patent 6,593,770</b></p>	<p><b>Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)</b></p>
	<p>Processor 22 then executes (step 50) the application software 40 resident in host memory 24 that corresponds to PCMCIA card 14. The application software 40 causes the processor to either select a default FPGA programming data file 42 from host memory 24 that corresponds to a particular application for PCMCIA card 14 or request input from the user as to which FPGA programming data file 42 is to be selected from host memory 24. Processor 22 then sends (step 52) the data from the selected FPGA programming data file 42 through PCMCIA adapter 16 to PCMCIA interface chip 26. Interface chip 26 then programs (step 54) a field programmable gate array (FPGA, not shown in FIG. 1) within card controller 32 by loading the data from the FPGA programming data file 42 into the FPGA. Where the application software causes the processor to select a default data programming file, PCMCIA card 14 and host computer 12 are made operable (step 56) without user intervention.</p> <p>Michelson, Ex. 1003, 6:61-65:</p> <p>The FPGA programming data files can be supplied with the PCMCIA card or new, additional, or updated FPGA programming data files can be obtained at a later time. Similarly, users can create their own FPGA programming data files or make modifications as desired.</p>
<p><b>Claim 10</b></p>	
<p>The system of claim 1, wherein said second circuit comprises a reset circuit configured to reset the first or second configuration of the peripheral</p>	<p style="text-align: center;"><b>MICHELSON</b></p> <p>Michelson, Ex. 1003, 4:9-22:</p> <p>Referring to FIG. 3, card controller 32 includes a PCMCIA card controller FPGA 60 (e.g., part number XC3042TQ100-100, manufactured by Xilinx, as described in Xilinx Programmable Logic Data Book, which is hereby incorporated by reference). Through a standard PCMCIA bus (i.e., PCMCIA address lines 62, data lines 64, and control lines 66) connected to PCMCIA connector 28, interface chip 26 receives FPGA programming data from host computer 12 (FIG. 1). Interface chip 26 initiates FPGA 60 programming through FPGA</p>

<b>U.S. Patent 6,593,770</b>	<b>Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)</b>
----------------------------------	------------------------------------------------------------------------------------------

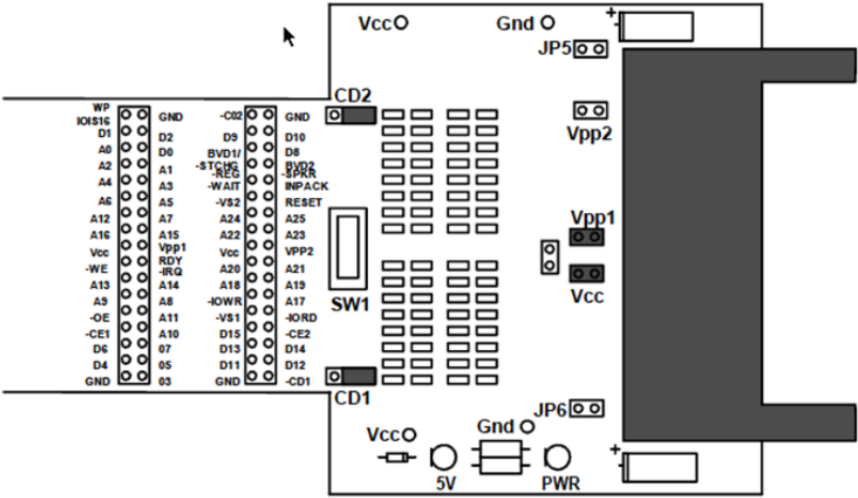
device.

programming circuit 68, which drives reset line 63 and reprogram line 65, and completes FPGA 60 programming by loading the FPGA programming data into FPGA 60 through peripheral data lines 72.

**PCCextend**

PCCextend, Ex. 1004, p. 1 (“Using the PCCextend is relatively straightforward. The extender card is inserted into the desired slot in the host system. Then the PC Card under test is inserted into the card connector.”).

PCCextend, Ex. 1004, p. 1 (“Caution: Insertion and removal of the extender and PC card should be done with care. The PC Card’s fragile connectors may be broken or bent if improper force is used. Both card and extender should be inserted straight without any lateral movement or force.” (Italics and bold omitted.)).

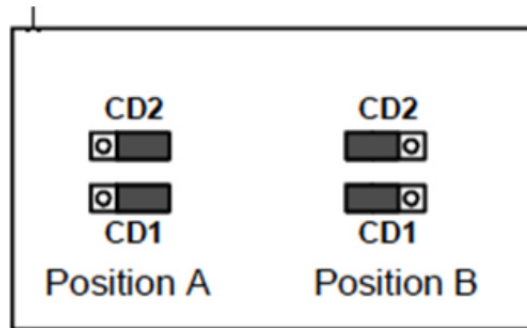


PCCextend, Ex. 1004, pp. 3-4:

**2.3 Using the PCCswitch**

PCCextend 100 includes the PCCswitch, which can be used to momentarily interrupt the CD1 and CD2 signals. The PCCswitch is centrally located on the PCCextend 100 between the termination area and test points. When properly configured, the PCCswitch can interrupt the card detect signals (-CD 1 and

-CD2) to simulate a card removal/insertion cycle. . . . When a card is inserted, CD1 and CD2 may be momentarily interrupted by pressing the PCCswitch.

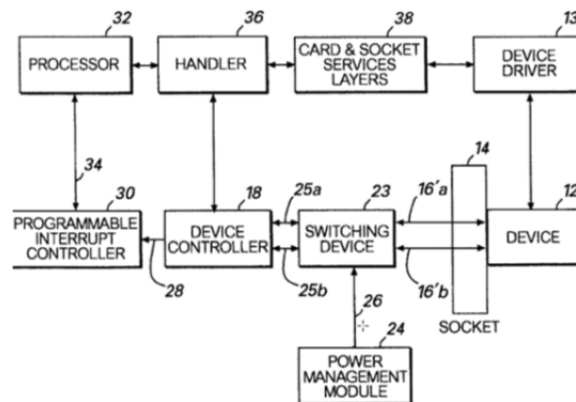


**Figure 2.3-1 Card Detect Switch**

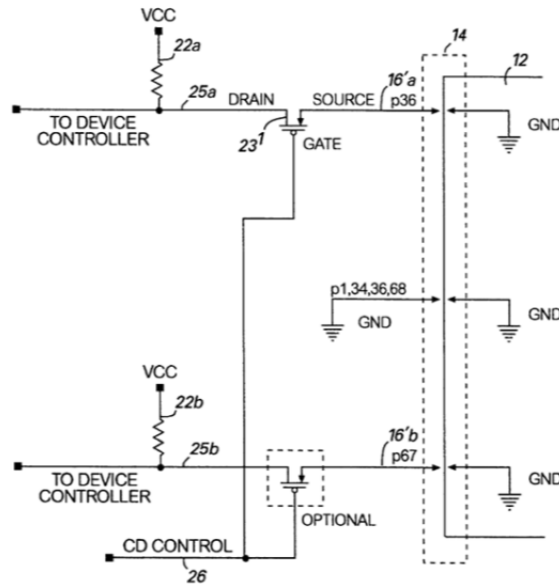
To test the operation of the PCCswitch, be sure that your PC Card Software drivers are loaded. Momentarily press the PCCtest switch. Most software drivers will issue a removal beep followed by an insertion beep. The software may also remove power from the socket when the card is removed.

**DAVIS**

Davis, Ex. 1005, Figs. 2 and 3:



**FIG. 2**



**FIG. 3**

Davis, Ex. 1005, 7:8-36:

FIG. 2 is a block diagram illustrating the components for an embodiment of the present invention. Referring now to FIG. 2, a switching device 23 is connected between the device 12 (via the socket 14) and the device controller 18. Specifically, the switching device 23 is connected to the socket 14 (and, if inserted, the device 12) by the card detect lines 16a' and 16b'. In addition, the switching device 23 is connected to the device controller 18 via the system advisory lines 25a and 25b. A power management module 24, which supports the power management function by controlling the power states of the computer and its devices, communicates with the switching device 23 via a control line 26. The control line 26 carries control signals output by the power management module 24 for controlling the operating state of the switching device 23. The switching device 23 can operate in the open state in response to a particular control signal from the power management module 24, thereby breaking a signal path between the card detect lines 16a' and 16b' and system advisory lines 25a and 25b. In the alternative, the switching device 23 can operate in the closed

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>state in response to another control signal to maintain a signal connection between the card detect lines 16a' and 16b' and the system advisory line 25a and 25b. It will be appreciated that the switching device 23 can be implemented by an electronic switch, typically a field effect transistor (FET) or a bipolar transistor. Other types of electronic switches, however, can be used to implement the switching device 23, as is known in the art.</p> <p>Davis, Ex. 1005, 10:21-35:</p> <p>FIG. 3 is a schematic diagram for the preferred embodiment of the present invention. Turning now to FIGS. 2 and 3, the switching device 23 is preferably implemented by a FET device 23' connected between either (1) the combination of the card detect line 1 6a and the system advisory line of 25a or (2) the combination of the card detect line 1 6b and the system advisory line 25b. The FET 23' serves to manipulate a signal control path between the socket 14 and the device controller 18, and a device removal event can be represented by deactivating the FET and opening this single path. Likewise, a device insertion event can be represented by activating the FET and closing this signal path (while the device 12 is properly installed within the socket 14 and a card detect line extends between the socket and the device controller).</p> <p>Davis, Ex. 1005, 8:31-34:</p> <p>By manipulating the operating state of the switching device 23, information regarding device insertion or removal events can be communicated to the card controller 18. 8:31-34.</p>
Claim 11	
[a] A method for reconfiguring a peripheral device having a first	<i>See discussion above with respect to claim element 1[a].</i>

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
configuration connected by a computer bus to a host computer, the method comprising the steps of:	
[b] (A) downloading information for a second configuration from the host computer into the peripheral device over the computer bus; and	<i>See discussion above with respect to claim element 1[b].</i>
[c] (B) electronically simulating a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device to said second configuration	<i>See discussion above with respect to claim element 1[c].</i>
[d] while supplying electrical power to said peripheral	<i>See discussion above with respect to claim element 1[d].</i>

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
device.	
<b>Claim 12</b>	
The method of claim 11, wherein said first configuration comprises a generic configuration assigned to the peripheral device and said second configuration comprises any one of a plurality of unique manufacturer configurations.	<i>See discussion above with respect to claim 2.</i>
<b>Claim 13</b>	
The method of claim 11, wherein step (A) comprises: reading an identification code from the peripheral device, and selecting said second configuration based on said identification code.	<i>See discussion above with respect to claim 3.</i>

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
Claim 16	
<p>The method of claim 11, wherein said information for the second configuration comprises (i) configuration data and (ii) an executable code.</p>	<p style="text-align: center;"><b>MICHELSON</b></p> <p>Michelson, Ex. 1003, 1:50-57:</p> <p style="padding-left: 40px;">The programmable architecture of FPGAs is provided through programmable logic blocks interconnected by a hierarchy of routing resources. The devices are customized by loading programming data into internal static memory cells. FPGA programming data are design-specific data that define the functional operation of the FPGA's internal blocks and their interconnections (e.g., the functional operation of the PCMCIA card controller and interface circuit).</p> <p>Michelson, Ex. 1003, 2:18-44:</p> <p style="padding-left: 40px;">In preferred embodiments, the host computer includes a PCMCIA adapter circuit coupled to a PCMCIA host socket which is mechanically and electrically connected to a PCMCIA card connector on the PCMCIA card. A PCMCIA interface circuit is connected to the PCMCIA card connector on the PCMCIA card. Using Card and Socket Services software stored in host memory, the host processor reads Card Information Structure (CIS) data from a memory device, such as an EEPROM, on the PCMCIA card and configures the host computer and PCMCIA card to operate together. Additionally, using application software stored in host memory, the processor selects an FPGA programming data file from host memory and sends data from the selected FPGA programming data file through the PCMCIA adapter circuit to the PCMCIA interface circuit. The PCMCIA interface circuit loads the data into a PCMCIA card controller FPGA to program the FPGA. When an error or a different user application is detected or when a user creates a new FPGA programming data file or modifies an existing FPGA programming data file, the processor is instructed to select another FPGA programming data file from host memory. The processor then sends data from the newly selected FPGA programming data file to the</p>



<p><b>U.S. Patent 6,593,770</b></p>	<p><b>Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)</b></p>
	<p>PCMCIA interface circuit, and the PCMCIA interface circuit loads the data into the PCMCIA card controller FPGA to reprogram the FPGA. The PCMCIA card may also have a functionality circuit that includes additional functional hardware specific to the function of the PCMCIA card.</p> <p><i>See also Michelson, Ex. 1003, Fig. 2, 3:59-4:8, 5:37-51, 6:10-20.</i></p>
<p><b>Claim 17</b></p>	
<p>The method of claim 11, wherein step (B) comprises electronically resetting the configuration of the peripheral device, controllable by the peripheral device.</p>	<p><i>See discussion above with respect to claim 10.</i></p>
<p><b>Claim 18</b></p>	
<p>[a] A system for reconfiguring a peripheral device having a configuration connected by a computer bus to a host computer, the system comprising:</p>	<p style="text-align: center;"><b>MICHELSON</b></p> <p>Michelson, Ex. 1003, 1:7-16:</p> <p style="padding-left: 40px;">This invention relates to programming and reprogramming the hardware configuration of a (PCMCIA) card. Personal computer memory card international association (PCMCIA) cards are computer cards that meet the minimum compliance requirements of the PCMCIA standard (e.g., release 2.1, which is hereby incorporated by reference). PCMCIA cards are typically used to add functionality or memory to a personal, portable, or desktop computer (i.e., host computer).</p> <p>Michelson, Ex. 1003, 1:28-36:</p> <p style="padding-left: 40px;">A typical PCMCIA card includes a standard PCMCIA</p>

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>connector connected to a PCMCIA interface circuit through a standard PCMCIA bus. The PCMCIA interface circuit operates according to the standard PCMCIA protocol to send data to and receive data from a host computer. The typical PCMCIA card also includes a PCMCIA card controller that sends data to and receives data from the PCMCIA interface circuit and controls the operation of the functional hardware on the card.”).</p> <p>Michelson, Ex. 1003, 2:3-16:</p> <p>In general, the invention includes a PCMCIA card having an FPGA based card controller that is programmed with FPGA programming data stored on a host computer through a standard PCMCIA bus. Storing FPGA programming data on the host computer allows a user access to a practically unlimited number of FPGA programming data files to program and reprogram the FPGA of the PCMCIA FPGA based card controller for different applications and permits a user to supplement, update, improve, or otherwise modify operation for existing applications. Additionally, storing the FPGA programming data files on the host computer saves valuable PCMCIA card real estate, reduces the amount of power required by the card during FPGA programming, and reduces the cost of the PCMCIA hardware.”).</p> <p>Michelson, Ex. 1003, 2:17-42:</p> <p>In preferred embodiments, the host computer includes a PCMCIA adapter circuit coupled to a PCMCIA host socket which is mechanically and electrically connected to a PCMCIA card connector on the PCMCIA card. A PCMCIA interface circuit is connected to the PCMCIA card connector on the PCMCIA card. Using Card and Socket Services software stored in host memory, the host processor reads Card Information Structure (CIS) data from a memory device, such as an EEPROM, on the PCMCIA card and configures the host computer and PCMCIA card to operate together. Additionally, using application software stored in host memory, the</p>

processor selects an FPGA programming data file from host memory and sends data from the selected FPGA programming data file through the PCMCIA adapter circuit to the PCMCIA interface circuit. The PCMCIA interface circuit loads the data into a PCMCIA card controller FPGA to program the FPGA. When an error or a different user application is detected or when a user creates a new FPGA programming data file or modifies an existing FPGA programming data file, the processor is instructed to select another FPGA programming data file from host memory. The processor then sends data from the newly selected FPGA programming data file to the PCMCIA interface circuit, and the PCMCIA interface circuit loads the data into the PCMCIA card controller FPGA to reprogram the FPGA.

Michelson, Ex. 1003, Fig. 1:

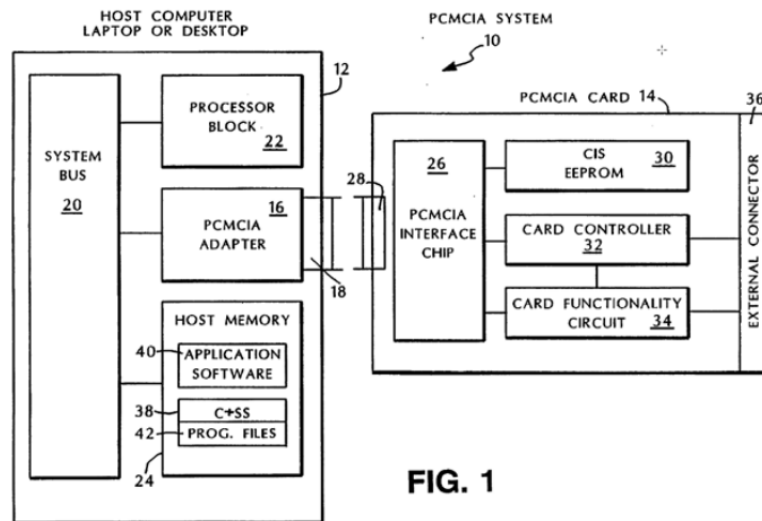


FIG. 1

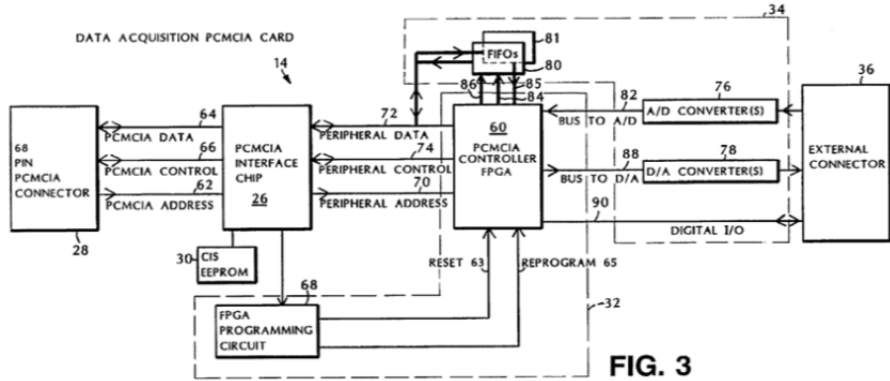
Michelson, Ex. 1003, 2:62-3:8:

Referring to FIG. 1, a PCMCIA system 10 includes a host computer 12 and a PCMCIA card 14. Within host computer 12, a PCMCIA adapter 16, connected to a standard 68 pin PCMCIA host socket 18 . . . , is coupled to a system bus 20 that interconnects PCMCIA adapter 16, a host processor 22, and a

<b>U.S. Patent 6,593,770</b>	<b>Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)</b>
----------------------------------	------------------------------------------------------------------------------------------

host memory 24. Within PCMCIA card 14, a PCMCIA interface chip 26 . . . , connected to a standard 68 pin PCMCIA card connector 28 . . . , is coupled to a Card Information Structure (CIS) EEPROM 30 . . . , a PCMCIA card controller 32, and a card functionality circuit 34.

Michelson, Ex. 1003, Fig. 3:



Michelson, Ex. 1003, 4:9-22

Referring to FIG. 3, card controller 32 includes a PCMCIA card controller FPGA 60 (e.g., part number XC3042TQ100-100, manufactured by Xilinx, as described in Xilinx Programmable Logic Data Book, which is hereby incorporated by reference). Through a standard PCMCIA bus (i.e., PCMCIA address lines data lines 64, and control lines 66) connected to PCMCIA connector 28, interface chip 26 receives FPGA programming data from host computer 12 (FIG. 1). Interface chip 26 initiates FPGA 60 programming through FPGA programming circuit 68, which drives reset line 63 and reprogram line 65, and completes FPGA 60 programming by loading the FPGA programming data into FPGA 60 through peripheral data lines 72.

[b] a first circuit configured to detect the peripheral device

**MICHELSON**  
Michelson, Ex. 1003, Fig. 2:

connected to the computer bus; and

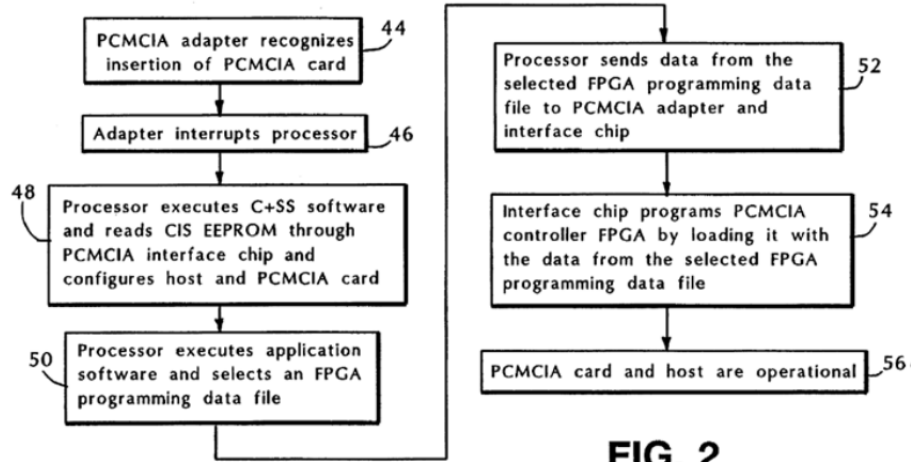


FIG. 2

Michelson, Ex. 1003, 2:34-58:

Referring also to FIG. 2, when PCMCIA card 14 card connector 28 is inserted in PCMCIA host socket 18 of host computer 12, PCMCIA adapter 16 recognizes (step 44) the insertion and interrupts (step 46) processor 22. Alternatively, if PCMCIA card 14 is inserted while host computer 12 is turned off (i.e., powered-down), host computer 12 learns of the existence of PCMCIA card 14 during the power-on procedure. Processor 22 then executes (step 48) Card and Socket Services 38 (C&SS) software resident in host memory 24 and through PCMCIA interface chip 26 reads CIS data from CIS EEPROM 30. As a minimum, the CIS data must sufficiently identify the PCMCIA card to the host, to enable the processor 22 to configure the host computer 12 and the PCMCIA card 14 to operate together and to enable the processor to select the appropriate application software 40 from host memory 24. The CIS data specifically identify the card manufacturer (e.g., Data Translation, Inc.) and card identification (ID) number and includes a variety of set-up information, including base address, interrupt level, size of address window, and other information regarding the card's functionality, as specified by release 2.1. The CIS data are entered into the EEPROM at the time of card manufacture and are not thereafter changed. Hence, configuration of host computer 12 and PCMCIA card

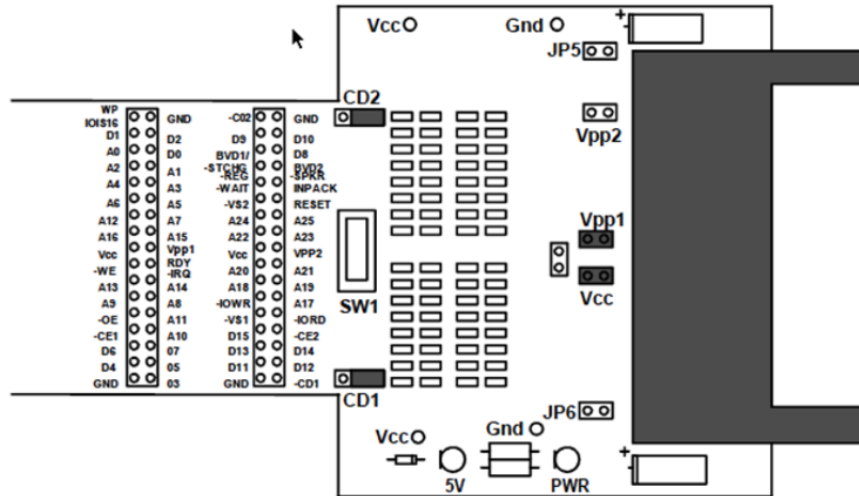
U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	14 is completed without the use of card controller 32.
<p>[c] a second circuit configured to electronically simulate a physical disconnection and reconnection of the peripheral device to reset said configuration of said peripheral device</p>	<p style="text-align: center;"><b>MICHELSON</b></p> <p>Michelson, Ex. 1003, 4:9-22:</p> <p>Referring to FIG. 3, card controller 32 includes a PCMCIA card controller FPGA 60 (e.g., part number XC3042TQ100-100, manufactured by Xilinx, as described in Xilinx Programmable Logic Data Book, which is hereby incorporated by reference). Through a standard PCMCIA bus (i.e., PCMCIA address lines 62, data lines 64, and control lines 66) connected to PCMCIA connector 28, interface chip 26 receives FPGA programming data from host computer 12 (FIG. 1). Interface chip 26 initiates FPGA 60 programming through FPGA programming circuit 68, which drives reset line 63 and reprogram line 65, and completes FPGA 60 programming by loading the FPGA programming data into FPGA 60 through peripheral data lines 72.</p> <p>Michelson, Ex. 1003, 5:37-5 1:</p> <p>For a variety of reasons, including the detection of a PCMCIA card controller 60 malfunction (e.g., error condition detected) or the detection of a user request to change the PCMCIA card application (e.g., software interrupt), PCMCIA card controller 60 may be reprogrammed. The host computer executes application software 40 (FIG. 1) in host memory 24 to select a new FPGA programming data file 42 from host memory 24 and then sends the data from the newly selected FPGA programming data file 42 through PCMCIA adapter 16 to PCMCIA interface chip 26. Interface chip 26 uses FPGA programming circuit 68 to reset PCMCIA card controller FPGA 60 and enable reprogramming, and interface chip 26 completes reprogramming by loading the data from the newly selected FPGA programming data file into card controller FPGA 60.</p> <p>Michelson, Ex. 1003, 5:52-6:60:</p>

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>For example, instead of simply storing all the data received from the sensors in the FIFO before sending it to the host computer, card controller FPGA 60 can be reprogrammed with additional functionality that formats the data received from the sensors according to the host computer's requirements before storing the data in the FIFO. Similarly, card controller FPGA 60 can be reprogrammed with additional functionality that analyzes the sensor data and interrupts the host computer when predetermined thresholds are exceeded. In such a system, the host computer need not interact with the PCMCIA card unless a predetermined threshold is exceeded.</p> <p>For example, a data acquisition PCMCIA card may be coupled to a temperature controller and sensors for determining the temperature of a room. Such a PCMCIA card continually receives data from the sensors and, in a simple data acquisition card, the host computer periodically reads the data acquisition FIFO and analyzes the data to determine if predetermined temperature thresholds have been exceeded. The PCMCIA card could be reprogrammed to analyze the data received from the sensors and interrupt the host computer when a predetermined temperature threshold is exceeded. Hence, the host computer would only read the PCMCIA card FIFO when notified that a threshold had been exceeded.</p> <p>Moreover, the operation of the PCMCIA card controller may need to change for different user applications. For example, if a temperature controller is moved to a smaller room where temperature fluctuates more quickly, the PCMCIA card controller needs to be reprogrammed to accept data from the temperature sensors more frequently. In such a situation, the user notifies the processor of a change in application and the processor selects a corresponding FPGA programming data file and sends the data to the PCMCIA interface which reprograms the PCMCIA card controller FPGA by loading the FPGA with the new FPGA programming data.</p> <p>Typically host memory is very large and is supplemented with extended memory (not shown). Hence, a practically unlimited number of FPGA programming data files can be stored within the host computer and made available to the user.</p>

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>The programming and reprogramming of the PCMCIA card controller FPGA is limited only by the size (i.e., capability) of the FPGA. The functionality of the PCMCIA card is limited only by the fixed hardware (i.e., the functional hardware) on the PCMCIA card (e.g., A/D or D/A converters, contacts on and configuration of the external connector, and FIFO size).</p> <p>For example, a PCMCIA card controller FPGA can be programmed to function as an I/O card controller, a data acquisition card controller, a fax/modem card controller, or a memory card controller; however, the PCMCIA card can only function as these card types if the additional functional hardware is available on the card. For an I/O card controller, the FPGA can be programmed (and reprogrammed) with the functionality required to transfer data between the host computer and the I/O bus (e.g., Small Computer System Interface (SCSI)), and, thus, the only additional functional hardware required is an external I/O bus connector and electrical conductors from the I/O bus connector to the FPGA. For a memory card, additional functional hardware typically includes a bank of static or dynamic RAM chips, ROM chips, flash memory, or a hard disk drive, and the addressing and refreshing functionality can be located within the PCMCIA card controller FPGA and, hence, reprogrammable. As an example, where a portion of a disk drive or a portion of a bank of RAMs becomes damaged and non-functional, the addressing functionality in the FPGA can be reprogrammed to address only the working portion of the memory hardware. For a fax/modem, additional functional hardware generally includes a phone connection, A/D and D/A converters, buffers, and amplifiers, and functionality controlling hardware, for instance, hardware controlling the baud rate, can be located within the FPGA and, thus, reprogrammable.</p> <p style="text-align: center;"><b>PCCextend</b></p> <p>PCCextend, Ex. 1004, p. 1 (“Using the PCCextend is relatively straightforward. The extender card is inserted into the desired slot in the host system. Then the PC Card under test is inserted into the card connector.”).</p>



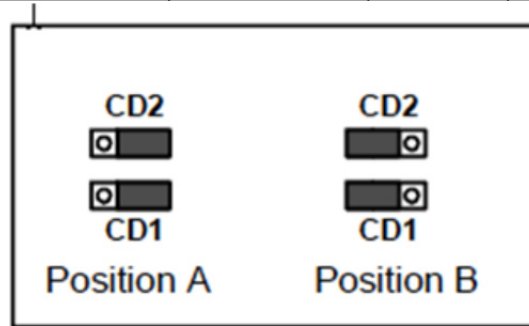
PCCextend, Ex. 1004, p. 1 (“Caution: Insertion and removal of the extender and PC card should be done with care. The PC Card’s fragile connectors may be broken or bent if improper force is used. Both card and extender should be inserted straight without any lateral movement or force.” (Italics and bold omitted.)).



PCCextend, Ex. 1004, pp. 3-4:

### 2.3 Using the PCCswitch

PCCextend 100 includes the PCCswitch, which can be used to momentarily interrupt the CD1 and CD2 signals. The PCCswitch is centrally located on the PCCextend 100 between the termination area and test points. When properly configured, the PCCswitch can interrupt the card detect signals (-CD 1 and -CD2) to simulate a card removal/insertion cycle. . . . When a card is inserted, CD1 and CD2 may be momentarily interrupted by pressing the PCCswitch.



**Figure 2.3-1 Card Detect Switch**

To test the operation of the PCCswitch, be sure that your PC Card Software drivers are loaded. Momentarily press the PCCtest switch. Most software drivers will issue a removal beep followed by an insertion beep. The software may also remove power from the socket when the card is removed.

**DAVIS**

Davis, Ex. 1005, 1:20-33 (“A power management system typically operates to conserve electrical power consumption by reducing power requirements in response to a detected lack of activity by a computer or its devices.”).

Davis, Ex. 1005, 1:43-49:

A power management event typically comprises either a power-down or power-up event. A sequence of power-down and power-up events can cause a computer device to enter a default state or a random state based on the loss of configuration information. It is often necessary to supply configuration information to a device via its device driver in response to a sequence of power-down and power-up events.

Davis, Ex. 1005, 2:21-32:

These removable devices can lose device configuration information in response to a power-down/power-up sequence in the absence of an appropriate power management system. Indeed, if the power management event is not communicated to

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>the device driver, the only way to return a device that has lost its device configuration to a useful state is to restart or reboot the computer system. The present invention solves these issues by using the device removal and insertion signals normally generated by the removal or insertion of a device to support a power management application and to advise a driver for a device about a power management event.</p> <p>Davis, Ex. 1005, 5:13-26:</p> <p>To achieve the desired power management function, it is often necessary to cause a removable device to enter a reduced power state when a device is inactive or placed in the idle state. However, a device will often lose its device configuration information in response to powering down that device. To fully use that device once power is restored, it is necessary to reconfigure the device with device configuration information. Specifically, it is desirable to communicate configuration information to a device via its device driver in response to restoring power. The present invention provides a solution to this problem by notifying the appropriate device driver of a power management event and by supplying device configuration information to a corresponding device in response to the restoration of power.</p> <p>Davis, Ex. 1005, 6:37-54:</p> <p>The state of the card detect lines 16a and 16b can be used by the device controller 18 to determine if a device 12 is connected to the socket 14. The device controller 18 determines that a device 12 is connected to the socket 14 when both card detect lines 16a and 16b transition from the logical high level to the logical low level. In contrast, in the event that one of the card detect lines 16a and 16b transitions from the logical low level to the logical high level, then the device controller 18 determines that a device 12 has been removed from the socket 14. In summary, a device insertion event is defined by a particular state for both of the card detect lines 16a</p>

and 16b, preferably each card detect line carrying a logical low level signal. A device removal event, however, is preferably defined by one of the card detect lines 16a and 16b transitioning to the logical high level. Those skilled in the art will appreciate that the removable device interface system described above is compatible with the standard specification for PCMCIA or PC Card devices.

Davis, Ex. 1005, Figs. 2 and 3:

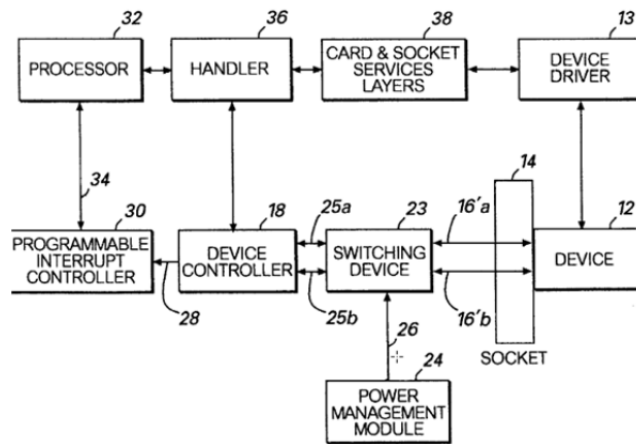
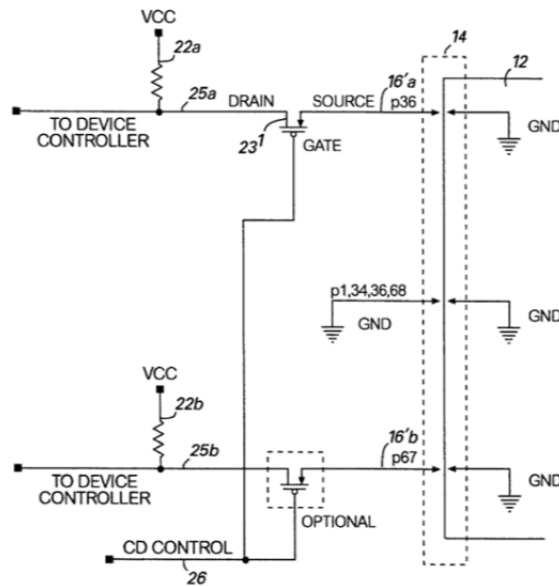


FIG. 2



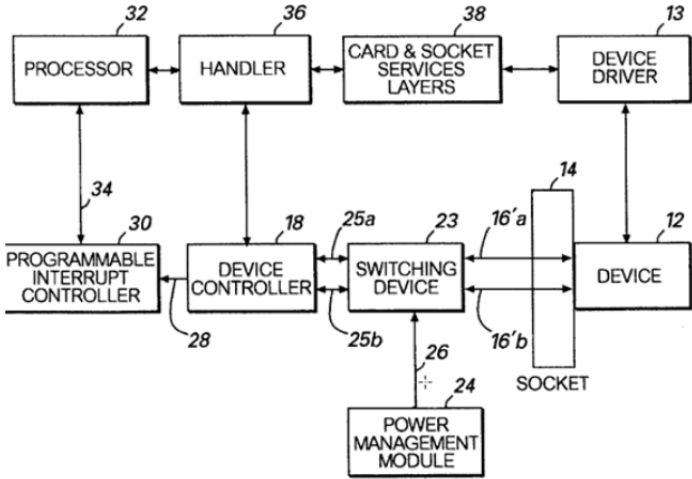
**FIG. 3**

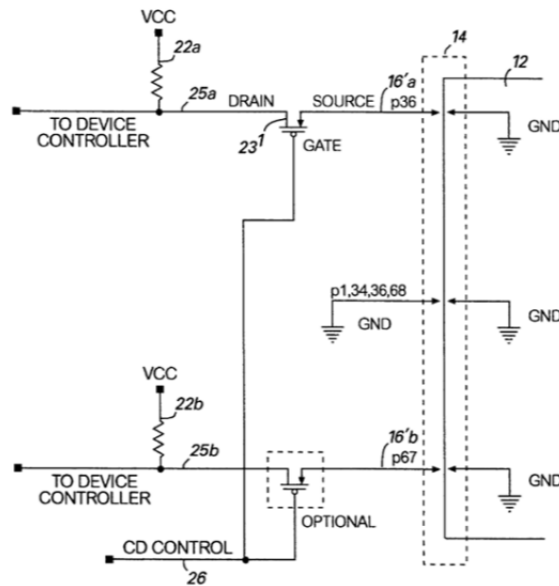
Davis, Ex. 1005, 7:8-36:

FIG. 2 is a block diagram illustrating the components for an embodiment of the present invention. Referring now to FIG. 2, a switching device 23 is connected between the device 12 (via the socket 14) and the device controller 18. Specifically, the switching device 23 is connected to the socket 14 (and, if inserted, the device 12) by the card detect lines 16a' and 16b'. In addition, the switching device 23 is connected to the device controller 18 via the system advisory lines 25a and 25b. A power management module 24, which supports the power management function by controlling the power states of the computer and its devices, communicates with the switching device 23 via a control line 26. The control line 26 carries control signals output by the power management module 24 for controlling the operating state of the switching device 23.

The switching device 23 can operate in the open state in response to a particular control signal from the power management module 24, thereby breaking a signal path between the card detect lines 16a' and 16b' and system advisory lines 25a and 25b. In the alternative, the switching device 23

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>can operate in the closed state in response to another control signal to maintain a signal connection between the card detect lines 16a' and 16b' and the system advisory line 25a and 25b. It will be appreciated that the switching device 23 can be implemented by an electronic switch, typically a field effect transistor (FET) or a bipolar transistor. Other types of electronic switches, however, can be used to implement the switching device 23, as is known in the art.</p> <p>Davis, Ex. 1005, 10:21-35:</p> <p>FIG. 3 is a schematic diagram for the preferred embodiment of the present invention. Turning now to FIGS. 2 and 3, the switching device 23 is preferably implemented by a FET device 23' connected between either (1) the combination of the card detect line 16a and the system advisory line of 25a or (2) the combination of the card detect line 16b and the system advisory line 25b. The FET 23' serves to manipulate a signal control path between the socket 14 and the device controller 18, and a device removal event can be represented by deactivating the FET and opening this single path. Likewise, a device insertion event can be represented by activating the FET and closing this signal path (while the device 12 is properly installed within the socket 14 and a card detect line extends between the socket and the device controller).</p> <p>Davis, Ex. 1005, 8:31-34:</p> <p>By manipulating the operating state of the switching device 23, information regarding device insertion or removal events can be communicated to the card controller 18. 8:31-34.</p> <p>Davis, Ex. 1005, 9:41-52:</p> <p>In response to a power-down event, the power management module outputs a particular control signal to the switching device 24, thereby causing the switching device to enter the open state or position. This interrupts the passage of signals from the card detect</p>

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	lines 16a' and 16b' to the device controller 18, thereby tricking the device controller to take actions in response to the apparent removal of the device 12. Significantly, the device 12 remains inserted within the socket 14, thereby leading to the presence of logical lower levels signals on the card detect lines 16a' and 16b' that represent a device insertion event.
[d] while supplying electrical power to said peripheral device.	See discussion above with respect to claim element 1[d].
Claim 20	
The system of claim 18, wherein said second circuit comprises a solid state switch.	<p style="text-align: center;"><b>DAVIS</b></p> <p style="text-align: center;">Davis, Ex. 1005, Figs. 2 and 3:</p>  <p style="text-align: center;"><b>FIG. 2</b></p> <p>The diagram, labeled FIG. 2, illustrates a system architecture. At the top, a horizontal chain of four blocks is connected by bidirectional arrows: PROCESSOR (32), HANDLER (36), CARD &amp; SOCKET SERVICES LAYERS (38), and DEVICE DRIVER (13). Below this chain, a PROGRAMMABLE INTERRUPT CONTROLLER (30) is connected to the PROCESSOR (32) via a bidirectional arrow (34). The PROGRAMMABLE INTERRUPT CONTROLLER (30) is also connected to a DEVICE CONTROLLER (18) via a bidirectional arrow (28). The DEVICE CONTROLLER (18) is connected to a SWITCHING DEVICE (23) via bidirectional arrows (25a and 25b). The SWITCHING DEVICE (23) is connected to a DEVICE (12) via bidirectional arrows (16'a and 16'b). The SWITCHING DEVICE (23) is also connected to a SOCKET (14) via bidirectional arrows (16'a and 16'b). A POWER MANAGEMENT MODULE (24) is connected to the SWITCHING DEVICE (23) via a bidirectional arrow (26).</p>



**FIG. 3**

Davis, Ex. 1005, 7:8-36:

FIG. 2 is a block diagram illustrating the components for an embodiment of the present invention. Referring now to FIG. 2, a switching device 23 is connected between the device 12 (via the socket 14) and the device controller 18. Specifically, the switching device 23 is connected to the socket 14 (and, if inserted, the device 12) by the card detect lines 16a' and 16b'. In addition, the switching device 23 is connected to the device controller 18 via the system advisory lines 25a and 25b. A power management module 24, which supports the power management function by controlling the power states of the computer and its devices, communicates with the switching device 23 via a control line 26. The control line 26 carries control signals output by the power management module 24 for controlling the operating state of the switching device 23.

The switching device 23 can operate in the open state in response to a particular control signal from the power management module 24, thereby breaking a signal path between the card detect lines 16a' and 16b' and system advisory lines 25a and 25b. In the alternative, the switching device 23



U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and Davis (Ex. 1005)
	<p>can operate in the closed state in response to another control signal to maintain a signal connection between the card detect lines 16a' and 16b' and the system advisory line 25a and 25b. It will be appreciated that the switching device 23 can be implemented by an electronic switch, typically a field effect transistor (FET) or a bipolar transistor. Other types of electronic switches, however, can be used to implement the switching device 23, as is known in the art.</p> <p>Davis, Ex. 1005, 10:21-35:</p> <p>FIG. 3 is a schematic diagram for the preferred embodiment of the present invention. Turning now to FIGS. 2 and 3, the switching device 23 is preferably implemented by a FET device 23' connected between either (1) the combination of the card detect line 16a and the system advisory line of 25a or (2) the combination of the card detect line 16b and the system advisory line 25b. The FET 23' serves to manipulate a signal control path between the socket 14 and the device controller 18, and a device removal event can be represented by deactivating the FET and opening this single path. Likewise, a device insertion event can be represented by activating the FET and closing this signal path (while the device 12 is properly installed within the socket 14 and a card detect line extends between the socket and the device controller).</p>

**D. Ground 4: Claims 5, 7, 15, and 19 are unpatentable under 35 U.S.C. § 103 as being obvious over Michelson (U.S. Patent No. 5,628,028), PCCextend, Davis (U.S. Patent No. 5,862,393), and APA**

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and APA (Ex. 1001)
<b>Claim 5</b>	
<p>The system of claim 1, wherein said computer bus comprises a Universal Serial Bus.</p>	<p style="text-align: center;"><b>APA</b></p> <p>APA, Ex. 1001, 2:20-39:</p> <p style="padding-left: 40px;">In a serial bus system, such as the USB, the only opportunity for associating software device drivers with a peripheral device is at the time when the peripheral device is plugged into the USB and the enumeration process occurs. Thus, to alter the configuration or personality of a peripheral device, such as downloading new code or configuration information into the memory of the peripheral device, the host computer system must detect a peripheral device connection or a disconnection and then a reconnection.</p> <p style="text-align: center;">. . .</p> <p>Thus, there is a need for a system and method for interfacing to a universal serial bus which avoids these and other problems of known systems and methods, and it is to this end that the present invention is directed.</p>
<b>Claim 7</b>	
<p>The system of claim 5, wherein said information for said second configuration comprises (i) configuration data and (ii) an executable code.</p>	<p style="text-align: center;"><b>MICHELSON</b></p> <p>Michelson, Ex. 1003, 1:50-57:</p> <p style="padding-left: 40px;">The programmable architecture of FPGAs is provided through programmable logic blocks interconnected by a hierarchy of routing resources. The devices are customized by loading programming data into internal static memory cells. FPGA programming data are design-specific data that define the functional operation of the FPGA's internal blocks and their interconnections (e.g., the functional operation of the PCMCIA card controller and interface circuit).</p>

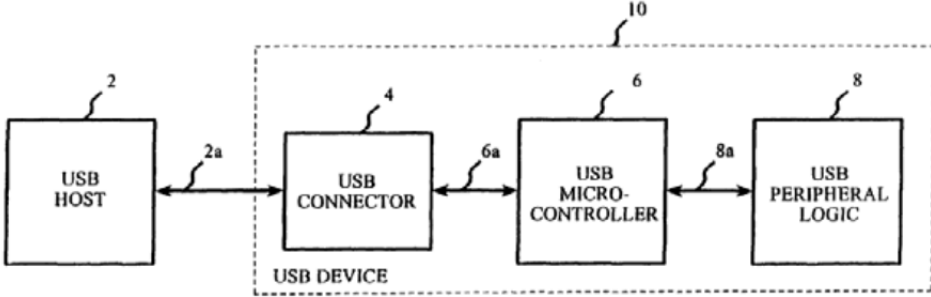
U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and APA (Ex. 1001)
	<p>Michelson, Ex. 1003, 2:18-44:</p> <p>In preferred embodiments, the host computer includes a PCMCIA adapter circuit coupled to a PCMCIA host socket which is mechanically and electrically connected to a PCMCIA card connector on the PCMCIA card. A PCMCIA interface circuit is connected to the PCMCIA card connector on the PCMCIA card. Using Card and Socket Services software stored in host memory, the host processor reads Card Information Structure (CIS) data from a memory device, such as an EEPROM, on the PCMCIA card and configures the host computer and PCMCIA card to operate together. Additionally, using application software stored in host memory, the processor selects an FPGA programming data file from host memory and sends data from the selected FPGA programming data file through the PCMCIA adapter circuit to the PCMCIA interface circuit. The PCMCIA interface circuit loads the data into a PCMCIA card controller FPGA to program the FPGA. When an error or a different user application is detected or when a user creates a new FPGA programming data file or modifies an existing FPGA programming data file, the processor is instructed to select another FPGA programming data file from host memory. The processor then sends data from the newly selected FPGA programming data file to the PCMCIA interface circuit, and the PCMCIA interface circuit loads the data into the PCMCIA card controller FPGA to reprogram the FPGA. The PCMCIA card may also have a functionality circuit that includes additional functional hardware specific to the function of the PCMCIA card.</p> <p><i>See also</i> Michelson, Ex. 1003, Fig. 2, 3:59-4:8, 5:37-51, 6:10-20.</p> <p style="text-align: center;"><b>APA</b></p> <p>APA, Ex. 1001, 2:20-39:</p> <p>In a serial bus system, such as the USB, the only opportunity for associating software device drivers with a</p>

U.S. Patent 6,593,770	Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and APA (Ex. 1001)
	<p>peripheral device is at the time when the peripheral device is plugged into the USB and the enumeration process occurs. Thus, to alter the configuration or personality of a peripheral device, such as downloading new code or configuration information into the memory of the peripheral device, the host computer system must detect a peripheral device connection or a disconnection and then a reconnection.</p> <p>...</p> <p>Thus, there is a need for a system and method for interfacing to a universal serial bus which avoids these and other problems of known systems and methods, and it is to this end that the present invention is directed.</p>
<b>Claim 15</b>	
<p>The method of claim 11, wherein step (A) comprises communicating said information for the second configuration to the peripheral device using a Universal Serial Bus.</p>	<p>See discussion above with respect to claim 5.</p>
<b>Claim 19</b>	
<p>The system of claim 18, wherein said computer bus comprises a Universal Serial Bus.</p>	<p style="text-align: center;"><b>APA</b></p> <p>APA, Ex. 1001, 2:20-39:</p> <p>In a serial bus system, such as the USB, the only opportunity for associating software device drivers with a peripheral device is at the time when the peripheral device is plugged into the USB and the enumeration process occurs. Thus, to alter the configuration or personality of a peripheral device, such as downloading new code or configuration information into the memory of the peripheral device, the host</p>

<b>U.S. Patent 6,593,770</b>	<b>Correspondence to Michelson (Ex. 1003), PCCextend (Ex. 1004) and APA (Ex. 1001)</b>
	<p>computer system must detect a peripheral device connection or a disconnection and then a reconnection.</p> <p style="text-align: center;">. . .</p> <p>Thus, there is a need for a system and method for interfacing to a universal serial bus which avoids these and other problems of known systems and methods, and it is to this end that the present invention is directed.</p>

**E. Ground 5: Claims 18-20 are unpatentable under 35 U.S.C. § 102 as being anticipated by Yap**

<b>U.S. Patent 6,593,770</b>	<b>Correspondence to Yap (Ex. 1002)</b>
<b>Claim 18</b>	
<p>[a] A system for reconfiguring a peripheral device having a configuration connected by a computer bus to a host computer, the system comprising:</p>	<p><b>YAP</b></p> <p>Yap, Ex. 1002, 4:6-10:</p> <p>Accordingly, when the USB micro-controller 6 drives the I/O pin to an appropriate logic state, the D+ and D- data lines may be opened or shorted via switching devices S+ and S-. By disconnecting the D+ and D- data lines via switching devices S+ and S-, a physical removal of the USB device 10 may be simulated in order to allow the USB host to re-configure the USB device 10 during a brown out condition.</p> <p><i>See also</i> Yap, Ex. 1002, 1:58-62, 4:6-10, 5:55-6:3.</p> <p><i>See also</i> Yap, Ex. 1002, Fig. 1:</p>

U.S. Patent 6,593,770	Correspondence to Yap (Ex. 1002)
	
<p>[b] a first circuit configured to detect the peripheral device connected to the computer bus; and</p>	<p style="text-align: center;"><b>YAP</b></p> <p>Yap, Ex. 1002, 4:22-23:</p> <p>[T]he USB micro-controller 6 opens the data lines via the switching devices S+ and S- for a duration greater than 2.5 micro-seconds and then reconnects them again. This procedure, for example, emulates the disconnect and re-connect procedure as specified in the USB specification v1.0, page 116.</p> <p><i>See also</i> Yap, Ex. 1002, 1:58-62, 2:20-25, 5:53-6:3.</p>
<p>[c] a second circuit configured to electronically simulate a physical disconnection and reconnection of the peripheral device to reset said configuration of said peripheral device</p>	<p>Yap, Ex. 1002, 1:55-2:3:</p> <p>According to the USB specification v1.0, page 201, the host operating system is supposed to record the last error type without trying to re-establish communications with the noncommunicating USB device. When this occurs, (1) the user may have to re-boot the USB device or physically disconnect and then re-connect the USB device to allow the host computer to recognize and then re-configure the USB device, or (2) the host computer operating system must be smart enough to avoid terminating the USB device when the USB device is terminally busy not communicating (e.g., continuously returning nonacknowledge (NAK) signals) and reset and re-configure the USB device. The first method defeats the whole purpose of plug-and-play technology, and the second method requires additional USB host computer operating system overhead to keep track of and recover from the USB device brown out condition.</p>

U.S. Patent 6,593,770	Correspondence to Yap (Ex. 1002)
	<p data-bbox="467 285 831 321">Yap, Ex. 1002, 2:20-25:</p> <p data-bbox="516 369 1453 537">It is also an object of the present invention to provide a method and apparatus for recovering from a USB device brown out condition without a need to re-boot the USB device or physically disconnect and then re-connect the USB device.</p> <p data-bbox="467 583 971 619">Yap, Ex. 1002, 4:3-23 and Fig. 2:</p> <p data-bbox="516 667 1453 1346">Accordingly, when the USB micro-controller 6 drives the I/O pin to an appropriate logic state, the D+ and D- data lines may be opened or shorted via switching devices S+ and S-. By disconnecting the D+ and D- data lines via switching devices S+ and S-, a physical removal of the USB device 10 may be simulated in order to allow the USB host to re-configure the USB device 10 during a brown out condition. . . . Firmware in the USB micro-controller 6 keeps the data lines connected via switching devices S+ and S- during normal operation. However, when a brown out condition is detected, as will be described later, the USB micro-controller 6 opens the data lines via the switching devices S+ and S- for a duration greater than 2.5 micro-seconds and then reconnects them again. This procedure, for example, emulates the disconnect and re-connect procedure as specified in the USB specification v1.0, page 116.</p>

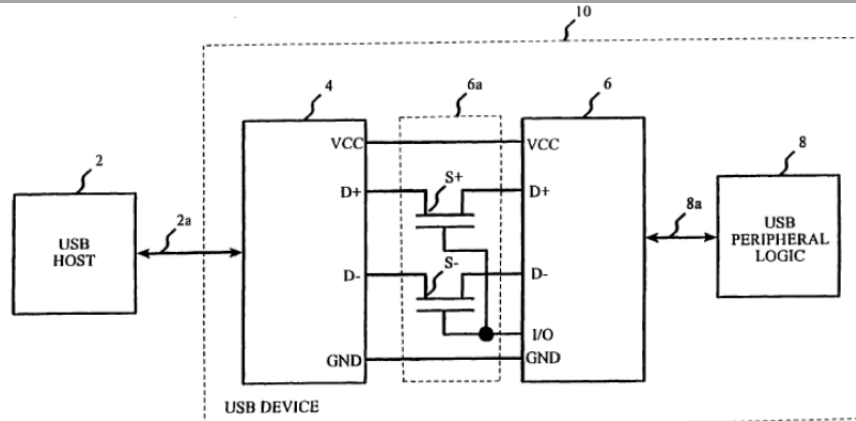


Fig. 2

Yap, Ex. 1002, 5:53-6:3:

[T]he data lines of the USB micro-controller 6 are opened via the switching devices S+ and S- for a duration greater than 2.5 micro-seconds and re-connected again. This procedure, for example, emulates the disconnect and re-connect procedure as specified in the USB specification v1.0, page 116, as previously discussed.

See also Yap, Ex. 1002, 1:40, 2:35, 2:46, 2:59, 3:60-63, 5:10-14, 5:56-6:3, 6:18-23 and Fig. 3.

[d] while supplying electrical power to said peripheral device.

**YAP**

Yap, Ex.1002, Figs. 2-3

Yap, Ex. 1002, 3:64-67:

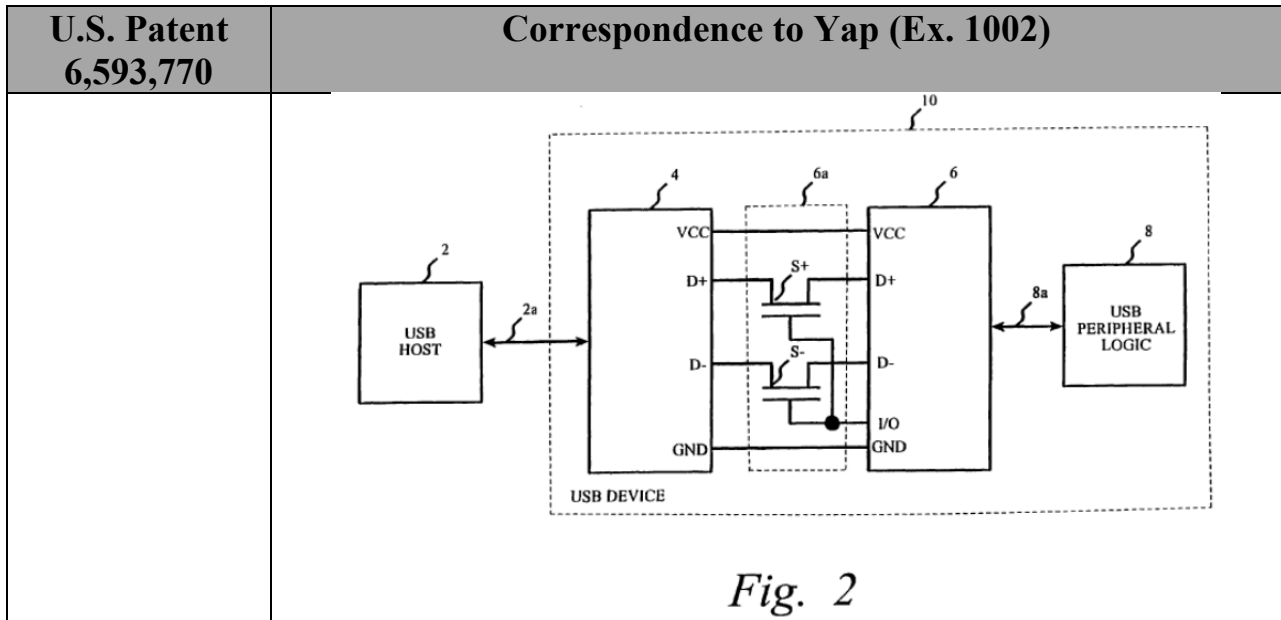
Please note that only one pair of complementary data lines of a plurality of complementary data lines and VCC and GND connections are shown in FIG. 2 for simplicity.

Yap, Ex. 1002, 4:10-16:

In addition, the general purpose I/O pin of the USB microcontroller 6 is configured such that during and after a



U.S. Patent 6,593,770	Correspondence to Yap (Ex. 1002)
	<p>reset condition due to power up the data lines stay connected (e.g., the I/O pin enables switching devices S+ and during and after reset). Firmware in the USB micro-controller 6 keeps the data lines connected via switching devices S+ and S- during normal operation.</p> <p>Yap, Ex. 1002, 4:3-23:</p> <p>Accordingly, when the USB micro-controller 6 drives the I/O pin to an appropriate logic state, the D+ and D- data lines may be opened or shorted via switching devices S+ and S-. By disconnecting the D+ and D- data lines via switching devices S+ and S-, a physical removal of the USB device 10 may be simulated in order to allow the USB host to re-configure the USB device 10 during a brown out condition.</p>
Claim 19	YAP
<p>The system of claim 18, wherein said computer bus comprises a Universal Serial Bus.</p>	<p>Yap, Ex. 1002, 3:60-63.</p> <p>In FIG. 2, a first embodiment of the USB device 10 of FIG. 1 further includes switching devices S+ and S-, such as transistors, contact switches, etc., coupled to positive data (D+) and negative data (D-) lines of the signal lines 6a.</p> <p>Yap, Ex. 1002, 4:15-23:</p> <p>However, when a brown out condition is detected, as will be described later, the USB micro-controller 6 opens the data lines via the switching devices S+ and S- for a duration greater than 2.5 micro-seconds and then reconnects them again. This procedure, for example, emulates the disconnect and re-connect procedure as specified in the USB specification v1.0, page 116.</p> <p><i>See also</i> Yap, Ex. 1002, 4:5-10, 4:24-30, 5:65-6:3 and Figs. 2-3:</p>



**F. Ground 6: Claim 11 is unpatentable under 35 U.S.C. § 102(b), as being anticipated over the USB Spec.**

'770 Patent Claim 11	USB Spec. (Ex. 1013)
<p>A method for reconfiguring a peripheral device having a first configuration connected by a computer bus to a host computer, the method comprising the steps of:</p>	<p>USB Spec., Ex. 1013, p. 29:</p> <p>USB devices present a standard USB interface in terms of their:</p> <ul style="list-style-type: none"> <li>• Comprehension of the USB protocol</li> <li>• Response to standard USB operations such as configuration and reset</li> <li>• Standard capability descriptive information</li> </ul> <p>USB Spec., Ex. 1013, p. 32:</p> <p>Since the USB allows USB devices to attach to or detach from the USB at any time, bus enumeration for this bus is an on-going activity.</p> <p>USB Spec., Ex. 1013, p. 28:</p>

4.1.1 Bus Topology

The Universal Serial Bus connects USB devices with the USB host. The USB physical interconnect is a tiered star topology. A hub is at the center of each star. Each wire segment is a point-to-point connection between the host and a hub or function, or a hub connected to another hub or function. Figure 4-1 illustrates the topology of the USB.

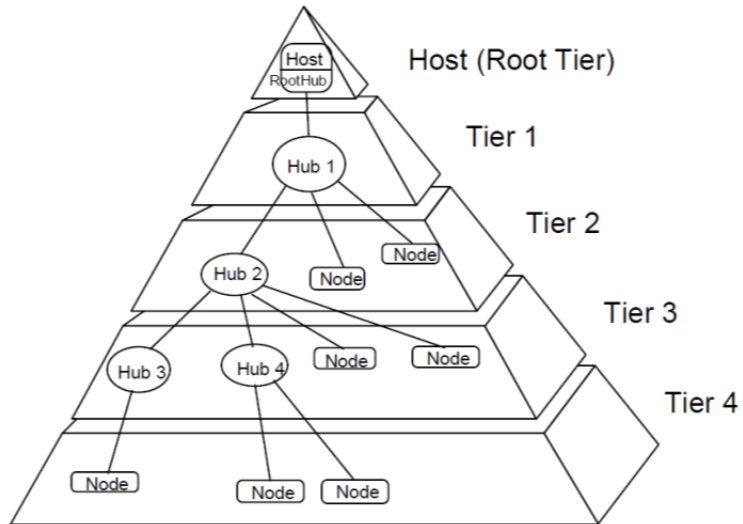


Figure 4-1. Bus Topology

USB Spec., Ex. 1013, p. 35:

Hubs are a key element in the plug-and-play architecture of USB. Figure 4-3 shows a typical hub. Hubs serve to simplify USB connectivity from the user's perspective and provide robustness at low cost and complexity.

Hubs are wiring concentrators and enable the multiple attachment characteristics of USB. Attachment points are referred to as ports. Each hub converts a single attachment point into multiple attachment points. The architecture supports concatenation of multiple hubs.

The upstream port of a hub connects the hub towards the host. Each of the other downstream ports of a hub allows connection to another hub or function. Hubs can detect attach and detach at each downstream port and enable the distribution of power to downstream devices. Each downstream port can be individually enabled and configured as either full or low speed. The hub isolates low speed ports from full speed signaling.

USB Spec., Ex. 1013, p. 34:

USB devices are divided into device classes such as hub, locator, or text device. The hub device class indicates a specially designated USB device which provides additional USB attachment points (refer to Chapter 11). USB devices are required to carry information for self-identification and generic configuration. They are also required at all times to display behavior consistent with defined USB device states.

USB Spec., Ex. 1013, p. 114:

7.1.3 Signal Termination

The USB is terminated at the hub and function ends as shown in Figure 7-5 and Figure 7-6. Full speed and low speed devices are differentiated by the position of the pull-up resistor on the downstream end of the cable. Full speed devices are terminated as shown in Figure 7-5 with the pull-up on the D+ line. Low speed devices are terminated as shown in Figure 7-6 with the pull-up on the D- line.

Low speed devices are terminated as shown in Figure 7-6 with the pull-up on the D- line. The pull-up terminator is a  $1.5\text{ k}\Omega \pm 5\%$  resistor tied to a voltage source between 3.0 V and 3.6 V referenced to the local ground. The pulldown terminators are resistors of  $15\text{ k}\Omega \pm 5\%$  connected to their local ground.

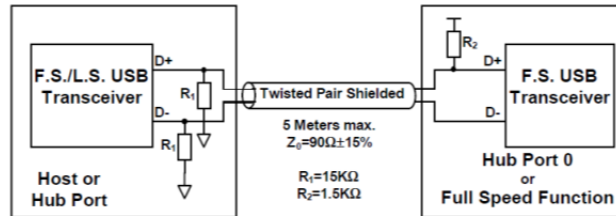


Figure 7-5. Full Speed Device Cable and Resistor Connections

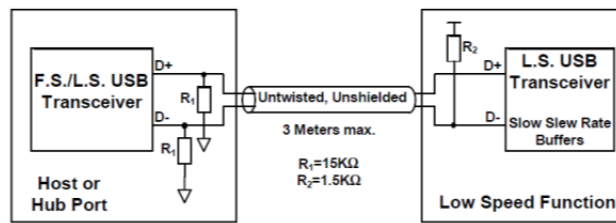


Figure 7-6. Low Speed Device Cable and Resistor Connections

USB Spec., Ex. 1013, p. 117:

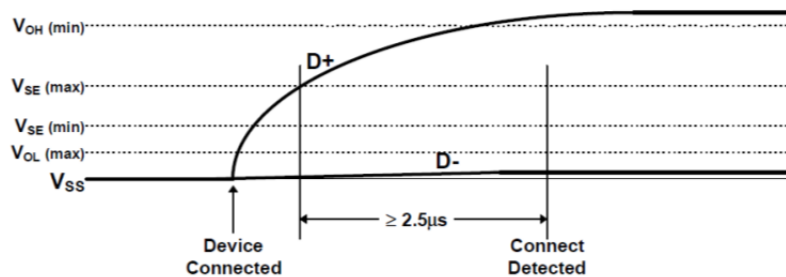


Figure 7-8. Full Speed Device Connect Detection

USB Spec., Ex. 1013, p. 117:

**'770 Patent  
Claim 11**

**USB Spec. (Ex. 1013)**

All ports on the downstream side of the host or a hub have pull-down resistors on both the D+ and D- lines. All devices have a pull-up resistor on one of the data lines on their upstream port. The type of device determines which data line has the pull-up resistor. Full speed devices have the pull-up on the D+ line (see Figure 7-5) and low speed devices have the pull-up on the D- line (see Figure 7-6). When a device is attached to hub or host but the data lines are not being driven, these resistors create a quiescent bias condition on the lines such that the data line with the pull-up is above 2.8 V and the other data line is near ground. This is called the idle state.

When no function is attached to the downstream port of the host or hub or the pull-up resistor on an attached device is not powered, the pull-down resistors will cause both D+ and D- to be pulled below the single-ended low threshold of the host or hub port. This creates a state called a single-ended zero (SE0) on the downstream port. A disconnect condition is indicated if an SE0 persists on a downstream port for more than 2.5  $\mu$ s (30 full speed bit times). Note that disconnect signaling applies only in an upstream direction (see Figure 7-7).

A connect condition will be detected when a device is connected to the host or hub's port, and one of the data lines is pulled above the single-ended high threshold level for more than 2.5  $\mu$ s (30 full speed data bit times). The data line that is high when the port state changes from disconnected to connected sets the idle state for this bus segment and determines whether the connected device is a full speed device or a low speed device. All signaling levels given in Table 7-1 are set for this network segment (and this segment alone) once the idle state is determined. Figure 7-8 shows a full speed device connection sequence and Figure 7-9 shows a low speed device connection sequence.

All hub ports start out in an implied disconnected state after power is applied to that port. If a device is connected to the port, the port goes through the connect sequence described above to detect the device type and set the port signaling characteristics (refer to Section 11.2.3).

**USB Spec., Ex. 1013, p. 224:**

Hubs must differentiate between full speed and low speed devices when a device is connected to the bus or at power-up. Hubs detect whether a device is full or low speed when the hub port transitions from its disconnected to its disabled state. Devices attached to a hub are determined to be either full speed or low speed by detecting which data line (D- or D+) is pulled high. Low speed devices pull D- high, and full speed devices pull D+ high. Full speed signaling must not be transmitted to low speed devices. Doing so could cause low speed devices to mistakenly respond to full speed signaling and create a bus conflict. Communication between the host and the hub controller are always done using full speed signaling.

**USB Spec., Ex. 1013, p. 224:**

Bus state evaluation is done at the end of the frame and is able to discriminate between the SE0, the differential 1 and 0 bus states. When no device is connected to a downstream hub port, its pull-down resistors pull both D+ and D- below  $V_{SE(min)}$ .

Connect/Disconnect detect can only be performed after  $V_{bus}$  is applied to the downstream port. (This requirement only affects hubs whose downstream ports are power switched.) When a device is connected, the bus state changes from the disconnected to the attach detect state. Low speed devices pull up D- to an SE1 and leave D+ at SE0. Full speed devices pull up D+ to an SE1 and leave D- at SE0. Each downstream hub port must be capable of detecting and differentiating between low speed and full speed device connections once a device is connected. The differential J and K states are undefined until a device is attached and the device's speed has been ascertained.

When a connect or disconnect occurs, it must be reflected in the hub status by the end of the frame in which the event occurred unless the hub is in the reset or suspend modes. A hub in the suspend mode is awakened by a connect or disconnect event and must be capable of reporting the event upon completion of resume. Upon coming out of reset, a hub must detect which downstream ports have devices connected to them. Connect and disconnect changes are reported on a per-port basis.

**USB Spec., Ex. 1013, p. 14:**

'770 Patent Claim 11	USB Spec. (Ex. 1013)
	<p><b>Default Address</b>      An address defined by the Universal Serial Bus Specification and used by a Universal Serial Bus device when it is first powered or reset. The default address is 00h.</p> <p>USB Spec., Ex. 1013, p. 168</p> <p>All USB devices use the default address when initially powered or after the device has been reset. Each USB device is assigned a unique address by the host after attachment or after reset. A USB device maintains its assigned address while suspended.</p> <p>USB Spec., Ex. 1013, p. 116:</p> <p>A connect condition will be detected when a device is connected to the host or hub's port, and one of the data lines is pulled above the single-ended high threshold level for more than 2.5 (30 full speed data bit times).</p> <p>USB Spec., Ex. 1013, p. 169:</p> <p>When a USB device is attached, the following actions are undertaken: ... 7. The host reads the configuration information from the device by reading each configuration 0 to n.</p>
(A) downloading information for a second configuration from the host computer into the peripheral device over the computer bus; and	<p>USB Spec., Ex. 1013, p. 14:</p> <p><b>Configuring Software</b>      The host software responsible for configuring a Universal Serial Bus device. This may be a system configurator or software specific to the device.</p> <p>USB Spec., Ex. 1013, p. 16:</p> <p><b>Device Software</b>      Software that is responsible for using a Universal Serial Bus device. This software may or may not also be responsible for configuring the device for use.</p> <p>USB Spec., Ex. 1013, p. 168:</p> <p>Before the USB device's function may be used, the device must be configured. From the device's perspective, configuration involves writing a non-zero value to the device configuration register. Configuring a device or changing an alternate setting causes all of the status and configuration values associated with endpoints in the affected interfaces to be set to their default values. This includes setting the data toggle of any endpoint using data toggles to the value DATA0.</p>

**'770 Patent  
Claim 11**

**USB Spec. (Ex. 1013)**

USB Spec., Ex. 1013, p. 169:

When a USB device is attached to or removed from the USB, the host uses a process known as bus enumeration to identify and manage the device state changes necessary. When a USB device is attached, the following actions are undertaken:

1. The hub to which the USB device is now attached informs the host of the event via a reply on its status change pipe (refer to Chapter 11 for more information). At this point, the USB device is in the attached state and the port to which it is attached is disabled.
2. The host determines the exact nature of the change by querying the hub.
3. Now that the host knows the port to which the new device has been attached, the host issues a port enable and reset command to that port.
4. The hub maintains the reset signal to that port for 10 ms. When the reset signal is released, the port has been enabled and the hub provides 100 mA of bus power to the USB device. The USB device is now in the powered state. All of its registers and state have been reset and it answers to the default address.
5. Before the USB device receives a unique address, its default pipe is still accessible via the default address. The host reads the device descriptor to determine what actual maximum data payload size this USB device's default pipe can use.
6. The host assigns a unique address to the USB device, moving the device to the addressed state.
7. The host reads the configuration information from the device by reading each configuration zero to n. This process may take several frames to complete.
8. Based on the configuration information and how the USB device will be used, the host assigns a configuration value to the device. The device is now in the configured state and all of the endpoints in this configuration have taken on their described characteristics. The USB device may now draw the amount of Vbus power described in its configuration descriptor. From the device's point of view it is now ready for use.

When the USB device is removed, the hub again sends a notification to the host. Detaching a device disables the port to which it had been attached. Upon receiving the detach notification, the host will update its local topological information.

USB Spec., Ex. 1013, p. 24:

- Self identifying peripherals, automatic mapping of function to driver, and configuration
- Dynamically attachable and reconfigurable peripherals

USB Spec., Ex. 1013, p. 24:

<b>PERFORMANCE</b>	<b>APPLICATIONS</b>	<b>ATTRIBUTES</b>
<b>LOW SPEED</b> •Interactive Devices •10-100 Kb/s	Keyboard, Mouse Stylus Game peripherals Virtual Reality peripherals Monitor Configuration	Lower cost Hot plug-unplug Ease of use Multiple peripherals
<b>MEDIUM SPEED</b> •Phone, Audio, Compressed Video 500Kb/s - 10Mbps	ISDN PBX POTS Audio	Low cost Ease of use Guaranteed latency Guaranteed Bandwidth Dynamic Attach-Detach Multiple devices
<b>HIGH SPEED</b> •Video, Disk •25-500 Mb/s	Video Disk	High Bandwidth Guaranteed latency Ease of use

Figure 3-1. Application Space Taxonomy

USB Spec., Ex. 1013, p. 34:

USB devices are divided into device classes such as hub, locator, or text device. The hub device class indicates a specially designated USB device which provides additional USB attachment points (refer to Chapter 11). USB devices are required to carry information for self-identification and generic configuration. They are also required at all times to display behavior consistent with defined USB device states.

USB Spec., Ex. 1013, p. 184:

A USB device has one or more configuration descriptors. Each configuration has one or more interfaces and each interface has one or more endpoints. An endpoint is not shared among interfaces within a single configuration unless the endpoint is used by alternate settings of the same interface. Endpoints may be shared among interfaces that are part of different configurations without this restriction.

Once configured, devices may support limited adjustments to the configuration. If a particular interface has alternate settings, an alternate may be selected after configuration. Within an interface, an isochronous endpoint's maximum packet size may also be adjusted.

USB Spec., Ex. 1013, p. 32:

Since the USB allows USB devices to attach to or detach from the USB at any time, bus enumeration for this bus is an on-going activity.

USB Spec. Ex. 1013, p. 202:



'770 Patent Claim 11	USB Spec. (Ex. 1013)									
	<p>Different operating system environments perform device configuration using different software components and different sequences of events. A specific operating system method is not assumed by the USB system. However, there are some basic requirements that must be fulfilled by any USB system implementation. In some operating systems, these requirements are met by existing host software. In others, the USB system provides the capabilities.</p> <p>The USB system assumes a specialized client of the USB, called a hub driver, which acts as a clearing house for addition of devices to and removal of devices from a particular hub. Once the hub driver receives such notifications, it will employ additional host software and other USB clients, in an operating system specific manner, to recognize and configure the device. This model, shown in Figure 10-4 is the basis of the following discussion.</p> <p>USB Spec., Ex. 1013, p. 214:</p> <p>Configuration Management services allow configuring software to replace a USB device configuration with another configuration from the set of configurations listed in the device. The operation succeeds if the data transfer rates given for all end points in the new configuration fit within the capabilities of the USB with the current schedule. If the new configuration is rejected, the previous configuration remains.</p>									
<p>(B) electronically simulating a physical disconnection and reconnection of the peripheral device to reconfigure the peripheral device to said second configuration while supplying electrical power to said peripheral device.</p>	<p>USB Spec., Ex. 1013, p. 32:</p> <p>Since the USB allows USB devices to attach to or detach from the USB at any time, bus enumeration for this bus is an on-going activity.</p> <p>USB Spec., Ex. 1013, p. 115:</p> <table border="1" data-bbox="621 1146 1408 1360"> <tbody> <tr> <td>Disconnect (Upstream only)</td> <td>(n.a.)</td> <td>D+ and D- &lt; <math>V_{se}</math>(max) for <math>\geq 2.5 \mu s</math></td> </tr> <tr> <td>Connect (Upstream only)</td> <td>(n.a.)</td> <td>D+ or D- &gt; <math>V_{se}</math>(max) for <math>\geq 2.5 \mu s</math></td> </tr> <tr> <td>Reset (Downstream only)</td> <td>D+ and D- &lt; <math>V_{se}</math> for <math>\geq 10 ms</math></td> <td>D+ and D- &lt; <math>V_{se}</math> (min) for <math>\geq 2.5 \mu s</math> (must be recognized within <math>5.5 \mu s</math>)<sup>3</sup></td> </tr> </tbody> </table> <p>USB Spec., Ex. 1013, p. 165:</p> <p>This section describes USB device states that are externally visible (see Figure 9-1). Note: USB devices perform a reset operation in response to a Reset request to the upstream port from the host. When reset signaling has completed, the USB device is reset.</p> <p>USB Spec., Ex. 1013, p. 116:</p>	Disconnect (Upstream only)	(n.a.)	D+ and D- < $V_{se}$ (max) for $\geq 2.5 \mu s$	Connect (Upstream only)	(n.a.)	D+ or D- > $V_{se}$ (max) for $\geq 2.5 \mu s$	Reset (Downstream only)	D+ and D- < $V_{se}$ for $\geq 10 ms$	D+ and D- < $V_{se}$ (min) for $\geq 2.5 \mu s$ (must be recognized within $5.5 \mu s$ ) <sup>3</sup>
Disconnect (Upstream only)	(n.a.)	D+ and D- < $V_{se}$ (max) for $\geq 2.5 \mu s$								
Connect (Upstream only)	(n.a.)	D+ or D- > $V_{se}$ (max) for $\geq 2.5 \mu s$								
Reset (Downstream only)	D+ and D- < $V_{se}$ for $\geq 10 ms$	D+ and D- < $V_{se}$ (min) for $\geq 2.5 \mu s$ (must be recognized within $5.5 \mu s$ ) <sup>3</sup>								

**'770 Patent  
Claim 11**

**USB Spec. (Ex. 1013)**

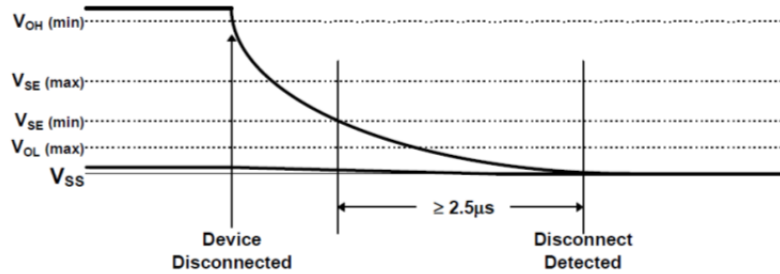
All ports on the downstream side of the host or a hub have pull-down resistors on both the D+ and D- lines. All devices have a pull-up resistor on one of the data lines on their upstream port. The type of device determines which data line has the pull-up resistor. Full speed devices have the pull-up on the D+ line (see Figure 7-5) and low speed devices have the pull-up on the D- line (see Figure 7-6). When a device is attached to hub or host but the data lines are not being driven, these resistors create a quiescent bias condition on the lines such that the data line with the pull-up is above 2.8 V and the other data line is near ground. This is called the idle state.

When no function is attached to the downstream port of the host or hub or the pull-up resistor on an attached device is not powered, the pull-down resistors will cause both D+ and D- to be pulled below the single-ended low threshold of the host or hub port. This creates a state called a single-ended zero (SE0) on the downstream port. A disconnect condition is indicated if an SE0 persists on a downstream port for more than 2.5  $\mu$ s (30 full speed bit times). Note that disconnect signaling applies only in an upstream direction (see Figure 7-7).

A connect condition will be detected when a device is connected to the host or hub's port, and one of the data lines is pulled above the single-ended high threshold level for more than 2.5  $\mu$ s (30 full speed data bit times). The data line that is high when the port state changes from disconnected to connected sets the idle state for this bus segment and determines whether the connected device is a full speed device or a low speed device. All signaling levels given in Table 7-1 are set for this network segment (and this segment alone) once the idle state is determined. Figure 7-8 shows a full speed device connection sequence and Figure 7-9 shows a low speed device connection sequence.

All hub ports start out in an implied disconnected state after power is applied to that port. If a device is connected to the port, the port goes through the connect sequence described above to detect the device type and set the port signaling characteristics (refer to Section 11.2.3).

**USB Spec., Ex. 1013, p. 117:**



**Figure 7-7. Disconnect Detection**

**USB Spec., Ex. 1013, p. 119:**

A reset is signaled downstream from a hub port on the bus by the presence of an extended SE0 at the upstream port of a device. After the reset is removed, the device will be in the attached, but not yet addressed or configured state (refer to Section 9.1). Note that reset signaling applies only in the downstream direction.

The reset signal can be generated by host command on any hub or host controller port. The reset signal must be generated for a minimum of 10 ms. The port that generated the reset will be sent to the logically disconnected state at the end of the reset. If a device is connected to the port, the bus pull-up resistor will determine the device type (low or full speed) and the port will end up in the disabled state (refer to Section 11.2.3).

An active device (powered and not in the suspend state) seeing a single-ended zero on its upstream port for more than 2.5  $\mu$ s may treat that signal as a reset, but must have interpreted the signaling as a reset within 5.5  $\mu$ s. A device that recognizes a reset from a SE0 between 32 and 64 full speed bit times or between 4 and 8 low speed bit times satisfies these requirements. The reset signal propagates through all enabled ports of any hubs downstream of the signaling port, but does not propagate through any ports that are disabled. A bus-powered hub that receives a reset on its root port removes power from all its downstream ports. After the reset is removed, all devices that received the reset are set to their default USB address and are in the unconfigured state. All ports on a hub that received a reset are disabled.

'770 Patent Claim 11	USB Spec. (Ex. 1013)
	<p>USB Spec., Ex. 1013, p. 224:</p> <p>Bus state evaluation is done at the end of the frame and is able to discriminate between the SE0, the differential 1 and 0 bus states. When no device is connected to a downstream hub port, its pull-down resistors pull both D+ and D- below <math>V_{SE(min)}</math>.</p> <p>Connect/Disconnect detect can only be performed after <math>V_{bus}</math> is applied to the downstream port. (This requirement only affects hubs whose downstream ports are power switched.) When a device is connected, the bus state changes from the disconnected to the attach detect state. Low speed devices pull up D- to an SE1 and leave D+ at SE0. Full speed devices pull up D+ to an SE1 and leave D- at SE0. Each downstream hub port must be capable of detecting and differentiating between low speed and full speed device connections once a device is connected. The differential J and K states are undefined until a device is attached and the device's speed has been ascertained.</p> <p>When a connect or disconnect occurs, it must be reflected in the hub status by the end of the frame in which the event occurred unless the hub is in the reset or suspend modes. A hub in the suspend mode is awakened by a connect or disconnect event and must be capable of reporting the event upon completion of resume. Upon coming out of reset, a hub must detect which downstream ports have devices connected to them. Connect and disconnect changes are reported on a per-port basis.</p> <p>USB Spec., Ex. 1013, p. 242:</p> <p>USB devices must power on in such a manner that they do not drive D+ or D- (except with the pull-up resistor) during the reset process. This is required so the upstream hub can drive reset downstream and be assured that the downstream device will see the reset signaling.</p> <p>USB Spec., Ex. 1013, p. 166:</p>

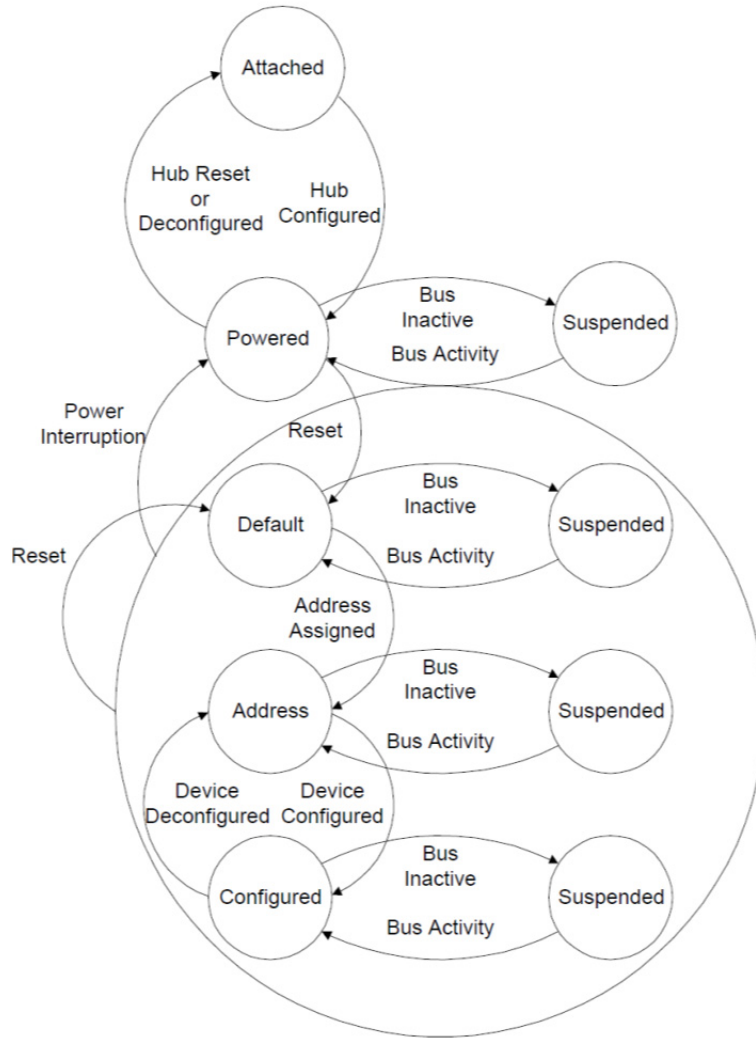


Figure 9-1. Device State Diagram

USB Spec., Ex. 1013, p. 167:

**'770 Patent  
Claim 11**

**USB Spec. (Ex. 1013)**

Table 9-1. Visible Device States

Attached	Powered	Default	Address	Configured	Suspended	State
No	--	--	--	--	--	Device is not attached to USB. Other attributes are not significant.
Yes	No	--	--	--	--	Device is attached to USB, but is not powered. Other attributes are not significant.
Yes	Yes	No	--	--	--	Device is attached to USB and powered, but has not been reset.
Yes	Yes	Yes	No	--	--	Device is attached to USB and powered and has been reset, but has not been assigned a unique address. Device responds at the default address.
Yes	Yes	Yes	Yes	No	--	Device is attached to USB, powered, has been reset, and a unique device address has been assigned. Device is not configured.
Yes	Yes	Yes	Yes	Yes	No	Device is attached to USB, powered, has been reset, has unique address, is configured, and is not suspended. Host may now use the function provided by the device.
Yes	Yes	Yes	Yes	Yes	Yes	Device is, at minimum, attached to USB, has been reset, and is powered at the minimum suspend level. It may also have a unique address and be configured for use. However, since the device is suspended, the host may not use the device's function.

USB Spec., Ex. 1013, p. 169:

**'770 Patent  
Claim 11**

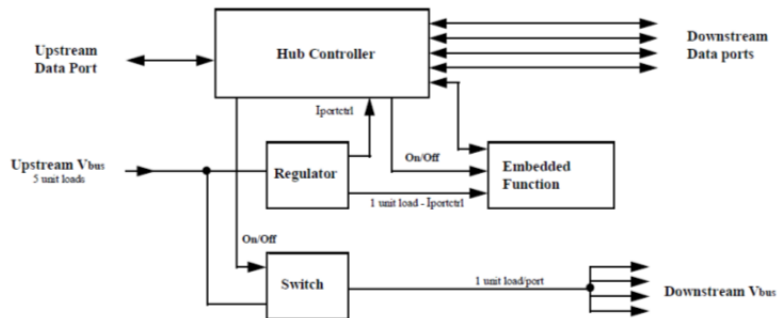
**USB Spec. (Ex. 1013)**

When a USB device is attached to or removed from the USB, the host uses a process known as bus enumeration to identify and manage the device state changes necessary. When a USB device is attached, the following actions are undertaken:

1. The hub to which the USB device is now attached informs the host of the event via a reply on its status change pipe (refer to Chapter 11 for more information). At this point, the USB device is in the attached state and the port to which it is attached is disabled.
2. The host determines the exact nature of the change by querying the hub.
3. Now that the host knows the port to which the new device has been attached, the host issues a port enable and reset command to that port.
4. The hub maintains the reset signal to that port for 10 ms. When the reset signal is released, the port has been enabled and the hub provides 100 mA of bus power to the USB device. The USB device is now in the powered state. All of its registers and state have been reset and it answers to the default address.
5. Before the USB device receives a unique address, its default pipe is still accessible via the default address. The host reads the device descriptor to determine what actual maximum data payload size this USB device's default pipe can use.
6. The host assigns a unique address to the USB device, moving the device to the addressed state.
7. The host reads the configuration information from the device by reading each configuration zero to n. This process may take several frames to complete.
8. Based on the configuration information and how the USB device will be used, the host assigns a configuration value to the device. The device is now in the configured state and all of the endpoints in this configuration have taken on their described characteristics. The USB device may now draw the amount of Vbus power described in its configuration descriptor. From the device's point of view it is now ready for use.

When the USB device is removed, the hub again sends a notification to the host. Detaching a device disables the port to which it had been attached. Upon receiving the detach notification, the host will update its local topological information.

USB Spec., Ex. 1013, p. 132:



**Figure 7-23. Compound Bus-powered Hub**

Power to downstream ports must be switched. The hub controller supplies a software controlled on/off signal from the host, which is in the "off" state when the device is powered up or after reset signaling. When switched to the "on" state, the switch implements a soft turn-on function which prevents excessive transient current from being drawn from the upstream port. The voltage drop across the upstream cable, connectors, and switch in a bus-powered hub must not exceed 350 mV at maximum rated current.