การออกแบบและจำลองของยูเอสบีโฮสคอนโทรเลอร์

Design and Simulation of Mini USB Host Controller

นาย ชูพันธุ์ ตันติธารา
นาย พลากร ยะโสภา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิศวกรรมอิเล็กทรอนิกส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2548

การออกแบบและจำลองของมินิยูเอสบีโฮสคอนโทรเลอร์

Design and Simulation of Mini USB Host Controller

โดย

นาย ชูพันธุ์ ตันติชารา

นาย พจากร ยะโสภา

อาจารย์ที่ปรึกษา

ดร. กสิน วิเชียรชม

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การออกแบบและจำลองของมินิยูเอสบีโฮสคอนโทรเลอร์

Design and Simulation of Mini USB Host Controller

ผู้จัดทำ

นาย ชูพันธุ์ ตันติธารา

นาย พลากร ยะโสภา

..... อาจารย์ที่ปรึกษา

(ดร. กสิน วิเชียรชม)

การออกแบบและจำลองมินิยูเอสบีโฮสคอนโทรเลอร์

นาย ชูพันธุ์ ตันติธารา 45010192
นาย พลากร ยะโสภา 45010518
ดร. กสิน วิเชียรชม อาจารย์ที่ปรึกษา

บทคัดย่อ

โครงงานนี้เป็นการศึกษาถึงการส่งผ่านข้อมูลระหว่างอุปกรณ์ USB 2 ตัว เพื่อที่จะพัฒนาวงจรเชื่อมต่อ (USB Host Controller Interfacing Board) ที่เป็นตัวกลางในการส่งผ่านข้อมูลระหว่างอุปกรณ์ทั้งสอง โดยมีขั้นตอนเริ่มจากการใช้ภาษา VHDL ในการออกแบบ และจำลองผลการทำงานบนคอมพิวเตอร์ด้วยโปรแกรม ไมโครซิม ผลการออกแบบและทดสอบ แสดงว่าระบบที่ออกแบบสามารถแยกและสร้างแพ็กเกจ (Packet) ตามมาตรฐาน USB ในโหมด บัลค์ (Bulk Mode) ด้วยความเร็วแบบ Full Speed12 หรือ Mbps ได้

Design and Simulation Mini USB Host Controller

Mr. Chupan Tantitara 45010192

Mr. Phlakorn Yasopa 45010518

Dr. Kasin Vichienchom Advisor

Education Year 2005

Abstract

This report describes a study and design of a Mini USB Host Controller. The controller enables a direct data transmission between two USB devices without using personal computer. The controller was designed in behavioral level using VHDL. The design was synthesized and simulated by a program named Microsim®. The simulations show that only some parts of the controller such as a packet assembly/disassembly module, a configuration module are working properly. The design can support the Bulk mode transmission at the full-speed i.e. 12 Mbps

กิตติกรรมประกาศ

ผู้จัดทำขอกราบขอบพระคุณ บิดา มารดา ที่ให้โอกาสในการศึกษา และขอขอบคุณ ดร.กสิน วิเชียรชม อาจารย์ที่ปรึกษา ที่คอยให้คำแนะนำและข้อคิดเห็นต่าง ๆ ในการทำงาน ช่วยให้ปริญญานิพนธ์นี้จะมีความสมบูรณ์ และช่วยให้ผู้จัดทำมีความรอบรู้ยิ่งขึ้น ขอขอบคุณอาจารย์ทุกๆท่านที่ได้ถ่ายทอดวิชาต่างๆช่วยให้มีความรู้พื้นฐานในการทำปริญญานิพนธ์เล่มนี้

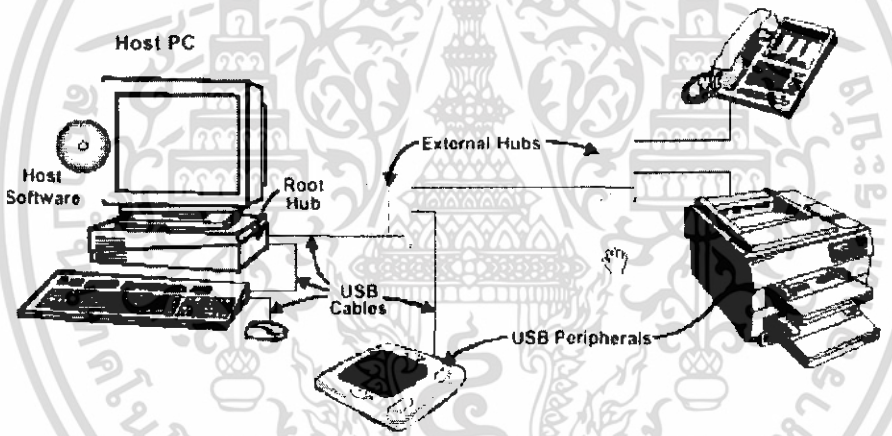
สารบัญ

สารบัญรูป

สารบัญตาราง

บทที่ 1
บทนำ

ในปัจจุบันอุปกรณ์เครื่องใช้ไฟฟ้าอิเลคทรอนิกส์จำนวนมากถูกออกแบบให้มีความสามารถในการเชื่อมต่อแลกเปลี่ยนข้อมูล และ ทำงานร่วมกับเครื่องคอมพิวเตอร์ได้ การส่งถ่ายข้อมูลระหว่างอุปกรณ์กับเครื่องคอมพิวเตอร์จึงได้รับการพัฒนาขึ้นอย่างต่อเนื่อง ซึ่งในปัจจุบัน หนึ่งทางเลือกในการติดต่อสื่อสารแลกเปลี่ยนข้อมูลกับเครื่องคอมพิวเตอร์ที่เป็นที่นิยมและมีการพัฒนาอย่างรวดเร็ว คงหนีไม่พ้น เทคโนโลยีการสื่อสารผ่านพอร์ท USB (Universal Serial Bus) โดยในปัจจุบันมีอุปกรณ์ต่อพ่วงที่ใช้การเชื่อมต่อแบบUSB จำนวนมาก อาทิ Flash drive memory , Printer ,scanner เป็นต้น ซึ่งอุปกรณ์เหล่านี้กำลังมีบทบาทในชีวิตประจำวัน ทั้งในด้านการเรียน การศึกษา การประกอบธุรกิจ การสื่อสาร และอื่นๆอีกมากมาย



รูปที่ 1.1 Example USB System Configuration

เพราะข้อดีๆหลายๆอย่างของเทคโนโลยีUSB และการพัฒนาอย่างต่อเนื่อง เป็นผลให้มีอุปกรณ์ต่อพ่วงเหล่านี้เกิดขึ้นจำนวนมาก จึงเกิดแนวคิดใหม่ขึ้นว่า เราจะทำอย่างไรให้อุปกรณ์ต่อพ่วงเหล่านั้นสามารถติดต่อและแลกเปลี่ยนข้อมูล หรือทำงานร่วมกัน โดยไม่ต้องผ่านเครื่องคอมพิวเตอร์ ซึ่งอาจเกินความจำเป็นในงานบางอย่าง โดยอุปกรณ์ต่อพ่วงUSB เหล่านี้จำเป็นต้องอาศัย โฮสคอนโทรลเลอร์ชนิดใหม่ที่ออกแบบมาเพื่อเป็นตัวกลางในการแลกเปลี่ยนข้อมูลระหว่างอุปกรณ์เหล่านั้น แทนเครื่องคอมพิวเตอร์ในการส่งถ่ายข้อมูลแบบเดิม

การออกแบบอุปกรณ์โฮสคอนโทรลเลอร์ที่จะนำมาใช้แทนเครื่องคอมพิวเตอร์นี้ จะออกแบบสร้างขึ้นโดยอิงตามมาตรฐานของUSB ที่ได้กำหนดไว้ โดยมีการออกแบบสร้างบนอุปกรณ์ FPGA และสร้างตัวดีไวซ์จำลอง(อุปกรณ์ต่อพ่วงUSB)ขึ้นด้วย เพื่อจำลองการส่งถ่ายข้อมูลเหมือนจริงที่เกิดขึ้น

บทที่ 2
ระบบยูเอสบี

ระบบบัสยูเอสบี เป็นระบบบัสที่ถูกออกแบบมาให้มีความง่ายในการเชื่อมต่อ และทรงประสิทธิภาพในการสื่อสารข้อมูลกับอุปกรณ์รอบข้างหลายๆ ชนิด โดยปราศจากข้อจำกัด และการขัดขวางของการอินเตอร์เฟสทางด้านฮาร์ดแวร์ ในโลกของยูเอสบี จะไม่มีการติดตั้งดิพสวิตซ์ หรือแม้แต่จั๊มเปอร์เพื่อการกำหนดค่าทางด้านฮาร์ดแวร์แต่อย่างใด ซึ่งการตั้งค่าการทำงานต่างๆ จะถูกกระทำโดยระบบปฏิบัติการอย่างอัตโนมัติ นอกจากนี้การเชื่อมต่ออุปกรณ์ยูเอสบี เข้ากับระบบยังสามารถทำในขณะที่เครื่องคอมพิวเตอร์ยังคงทำงานอยู่ได้ ซึ่งการใช้งานในลักษณะนี้คือรูปแบบของระบบปลั๊กแอนด์เพลย์ (Plug and Play) อย่างแท้จริง และในส่วนของการเพิ่มจำนวนพอร์ตการสื่อสารข้อมูลก็สามารถทำได้อย่างง่ายดายด้วยการใช้ยูเอสบีฮับ (USB Hub) มาต่อพ่วงเข้ากับระบบ จุดเด่นที่น่าสนใจอีกประการหนึ่งของระบบบัสชนิดนี้ก็คือ ความเร็วในการส่งถ่ายข้อมูลซึ่งสูงถึง 480 เมกะบิตต่อวินาทีสำหรับยูเอสบี ในเวอร์ชัน 2.0 ซึ่งสูงกว่าการเชื่อมต่อแบบขนานและอนุกรมเป็นอย่างมาก

จุดเด่นหลักๆ ของระบบบัส ยูเอสบี ได้แก่

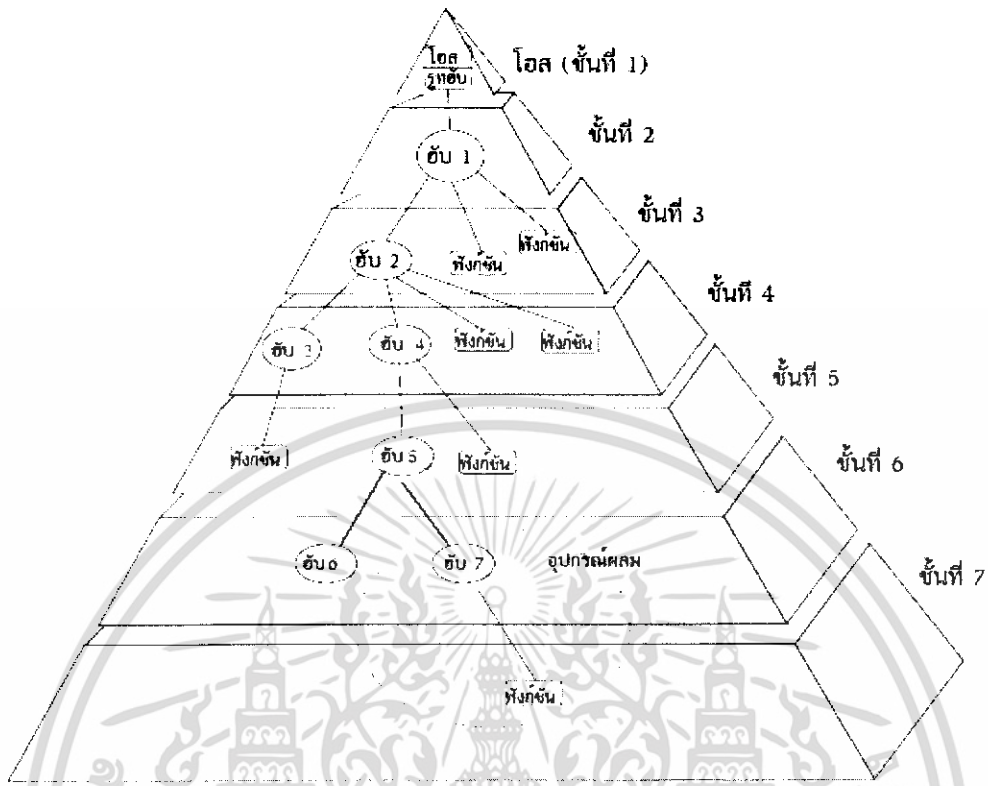
- ผู้ใช้สามารถนำอุปกรณ์ I/O มาต่อเข้ากับพอร์ตยูเอสบี ในขณะที่เครื่องคอมพิวเตอร์กำลังทำงานอยู่ได้
- ง่ายต่อการใช้งาน ซึ่งเครื่องคอมพิวเตอร์จะสามารถรู้จัก และจดจำอุปกรณ์ I/O ที่ถูกนำมาต่อเข้าไปในระบบโดยผ่านทางดีไวซ์ไดร์ฟเวอร์ที่เหมาะสมอีกทั้งการตั้งค่าต่างๆ จะเป็นไปอย่างอัตโนมัติ
- ลดความสับสนของการเชื่อมต่อด้วยการใช้คอนเนคเตอร์เพียงชนิดเดียว
- ประสิทธิภาพการส่งถ่ายข้อมูลสูงด้วยการส่งถ่ายข้อมูลชนิดเดียวที่มีความเร็วถึง 480 เมกะบิตวินาทีสำหรับอุปกรณ์แบบไฮสปีด ซึ่งมีค่าสูงกว่าพอร์ตขนานและอนุกรมหลายเท่า
- สามารถต่ออุปกรณ์ได้ถึง 127 ตัว
- สายเคเบิลที่ใช้เชื่อมต่อมจะมีสายของแหล่งจ่ายไฟรวมอยู่อยู่ภายในตัวมันด้วย
- มีการจัดการพลังงานที่ช่วยลดการใช้กำลังงานของตัวมันเองเมื่อไม่มีการใช้งาน
- มีการตรวจจับและแก้ไขความผิดพลาดของข้อมูลอย่างอัตโนมัติ

2.1 ความเป็นมาของยูเอสบี

พอร์ตแต่ละชนิดของคอมพิวเตอร์ได้รับการออกแบบมาเพื่องานเฉพาะ ทำให้อุปกรณ์แต่ละตัวต้องเลือกใช้พอร์ตต่างๆชนิดกันไป ส่งผลให้การนำอุปกรณ์ต่างๆมาเชื่อมต่อกับเครื่องคอมพิวเตอร์เป็นเรื่องที่ต้องให้ความสนใจ เนื่องจากอุปกรณ์แต่ละชนิดก็จะมีคอนเน็กเตอร์ที่ใช้เชื่อมต่อแตกต่างกันไปตามชนิดของพอร์ต เช่น พอร์ตอนุกรมที่ใช้ต่อคีย์บอร์ดและเมาส์ (ทั้งแบบ PS/2 และพอร์ตอนุกรมมาตรฐาน) พอร์ตขนานสำหรับต่อเครื่องพิมพ์พอร์ตเชื่อมต่อของจอภาพฯลฯทำให้ผู้ที่ใช้มีความรู้ในการติดตั้งหรือที่เรียกกันว่า เอ็นด์ยูสเซอร์(end user) พบกับความยากลำบากในการการเรียนรู้เรื่องราวเหล่านี้ นอกจากนั้นการติดตั้งอุปกรณ์ต่างๆเพิ่มเข้าไปในเครื่องคอมพิวเตอร์จะเกิดปัญหาการแย่งกันใช้สัญญาณ IRQ (Interrupt Request) ซึ่งเป็นตัวจำกัดจำนวนอุปกรณ์ที่จะมาต่อ ทำให้เกิดแนวความคิดที่จะกำหนดมาตรฐานเพื่อสร้างเป็นพอร์ตที่ทำให้การเชื่อมต่อทั้งหมดอยู่ในรูปแบบเดียวกันง่ายต่อการใช้งานของผู้ใช้ทั่วๆไป และไม่มีข้อจำกัดในการใช้ IRQ คำตอบของแนวคิดนี้คือ พอร์ตUSBนั่นเอง

2.2 โทโพโลยีของระบบบัส (Bus Topology)

โทโพโลยีหรือการจัดการเชื่อมต่อบนระบบบัส ยูเอสบี จะเป็นแบบไตรสตาร์ (Tiered Star) หรือสตาร์แบบลำดับชั้น (รูปที่ 2.1) โครงสร้างนี้ประกอบด้วยไปด้วยโครงสร้างแบบสตาร์หลายๆ ชุดเชื่อมต่อกันอยู่โดยมีฮับเป็นจุดศูนย์กลางของโครงสร้างระบบย่อยๆ เหล่านั้น ที่ส่วนบนสุดของสตาร์จะเป็นโฮสคอลโทรลเลอร์ซึ่งในระบบบัสยูเอสบี จะมีโฮสเพียงตัวเดียวเท่านั้นที่ทำหน้าที่ควบคุมการติดต่อสื่อสารทั้งหมดของระบบ และมีฮับเป็นตัวเพิ่มจำนวนของพอร์ตที่ใช้เชื่อมต่อกับอุปกรณ์รอบข้าง ทำให้ระบบสามารถเชื่อมต่ออุปกรณ์ได้มากขึ้น จุดปลายแต่จุดของโครงสร้างสตาร์จะเป็นอุปกรณ์รอข้างซึ่งต่อเข้ากับพอร์ตใดพอร์ตหนึ่งบนฮับหรืออาจนำมาเชื่อมต่อแทนอุปกรณ์เพื่อเพิ่มจำนวนพอร์ตให้มายิ่งขึ้น ซึ่งความสามารถในการเชื่อมต่อฮับเข้าด้วยกันจึงทำให้เกิดโครงสร้างย่อยได้หลายชุดและมีลักษณะเป็นลำดับชั้น



รูปที่ 2.1 โทโพโลยีของระบบบัส

การต่อแบบสตาร์ลำดับชั้นนั้นเป็นเพียงการอธิบายเพียงภาพเท่านั้น แต่ในความเป็นจริงนั้น ระบบบัสยูเอสบี จะมีเส้นทางข้อมูลสำหรับการสื่อสารเพียง 1 เส้นทางและโฮสกับอุปกรณ์ทุกตัวที่ต่ออยู่ในระบบจะส่งถ่ายข้อมูลเป็นลำดับชั้น

เนื่องจากตามปกติจะมีการหน่วงเวลาของสัญญาณเกิดขึ้นภายในเคเบิลและฮับ ดังนั้นจำนวนชั้นสูงสุดที่ยอมให้งานได้คือ 7 ชั้น (รวมชั้นที่เป็นรูทฮับด้วย) ภายในเจ็ดชั้นนี้มีจะมีฮับ (ไม่รวมรูทฮับ) ได้ไม่เกิน 5 ตัวและในชั้นที่ 7 จะต้องเป็นฟังก์ชันเท่านั้นในกรณีของอุปกรณ์ผสม นั้นด้วยโครงสร้างซึ่งมีฮับอยู่ภายในตัวมัน ทำให้อุปกรณ์นี้ครอบครองโครงสร้างไว้ 2 ชั้น ดังนั้นอุปกรณ์แบบผสมจึงไม่สามารถต่อในชั้นที่ 7 ได้ ซึ่งส่วนประกอบหลักของซอฟต์แวร์และฮาร์ดแวร์ที่จำเป็นสำหรับระบบ USB มีดังนี้

ส่วนซอฟต์แวร์

- ไดรเวอร์อุปกรณ์ USB (USB device driver)
- ไดรเวอร์ USB (USB driver)
- ไดรเวอร์โฮสคอนโทรลเลอร์ (USB host controller driver)

ส่วนฮาร์ดแวร์

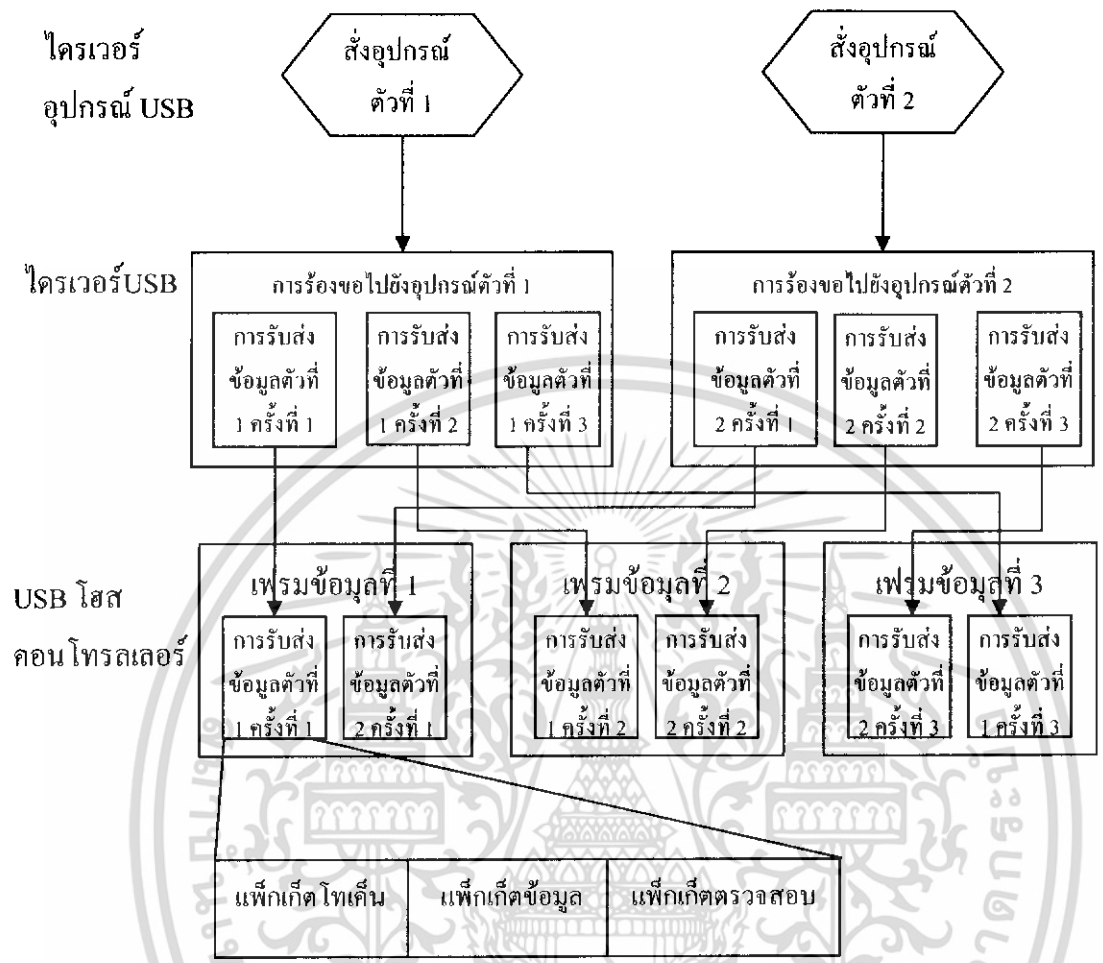
- USB โฮสคอนโทรลเลอร์ (USB host controller) / รูตฮับ (root hub)

- USB ฮับ (USB hub)
- อุปกรณ์ USB (USB device)

2.3 ส่วนประกอบทางซอฟต์แวร์

2.3.1 ไดรเวอร์อุปกรณ์ USB

ไดรเวอร์อุปกรณ์USB คือโปรแกรมเก็บข้อมูลที่จำเป็นในการติดต่อไปยังอุปกรณ์ แต่ละตัว เมื่อโปรแกรมใดมีความต้องการจะติดต่อกับอุปกรณ์ต่างๆ จะต้องแจ้งความต้องการนั้นๆ มายังไดรเวอร์อุปกรณ์ USB เนื่องจากตัวไดรเวอร์นี้จะรู้ว่าถ้าต้องการติดต่อกับอุปกรณ์จะต้องติดต่อผ่านเอ็นด์พอยต์ (Endpoint) ไหน ตัวรูปแบบใด (การทำงานของอุปกรณ์ USB จะติดต่อสั่งงานผ่านเอ็นด์พอยต์ของตัวอุปกรณ์ ซึ่งอุปกรณ์ต่างๆจะมีชนิดและจำนวนเอ็นด์พอยต์ที่ต่างกัน) ดังนั้นอุปกรณ์แต่ละตัวจะมีไดรเวอร์อุปกรณ์ USB เฉพาะตัว ซึ่งเมื่อถึงคราวต้องนำอุปกรณ์นั้นมาต่อใช้งานกับเครื่องคอมพิวเตอร์จริงๆก็จะต้องนำไดรเวอร์ตัวเดียวกันมาติดตั้งเพิ่มเข้ากับระบบปฏิบัติการในคอมพิวเตอร์เพื่อให้ระบบรู้จักและติดต่อใช้งานอุปกรณ์ที่ติดตั้งเข้ามาใหม่นี้ได้ เช่น ถ้าต้องการติดต่อเพื่อรับข้อมูลจากคีย์บอร์ด ตัวไดรเวอร์อุปกรณ์ USB จะรู้ว่าต้องรับส่งข้อมูลด้วยอัตราเร็วต่ำ(low speed) โดยใช้รูปแบบการถ่ายทอดข้อมูลแบบอินเตอร์รัปต์(Interrupt transfer type)ผ่าน เอ็นด์พอยต์ตัวหนึ่งของคีย์บอร์ด และตรวจสอบข้อมูลการกดเป็นช่วงระยะห่างค่าหนึ่ง



รูปที่ 2.2 ลำดับและขั้นตอนการทำงานของซอฟต์แวร์ควบคุมการทำงานของพอร์ตUSB

แต่ในบางอุปกรณ์ที่เป็นอุปกรณ์พื้นฐานของเครื่องคอมพิวเตอร์ เช่น เมาส์และคีย์บอร์ดจะมีการบรรจุไดรเวอร์อุปกรณ์ USBของอุปกรณ์เหล่านี้ไว้ภายในไบออสของเครื่องคอมพิวเตอร์เรียบร้อยแล้ว จึงไม่ต้องติดตั้งไดรเวอร์เพิ่มเติมสำหรับอุปกรณ์เหล่านี้ เพียงแต่เข้าไปเปิดการทำงาน ไบออสก็จะทำให้เครื่องคอมพิวเตอร์รู้จักอุปกรณ์เหล่านี้เอง

2.3.2 ไดรเวอร์ USB

การทำงานของUSB นั้นเป็นการต่อร่วมกันของอุปกรณ์หลายๆชนิดบนสายสัญญาณเพียงคู่เดียว ดังนั้นการส่งข้อมูลของอุปกรณ์แต่ละชนิดจะต้องมีการแบ่งสรรปันส่วนกันไปอย่างพอเหมาะพอดี เพื่อให้อุปกรณ์ทุกตัวสามารถทำงานทำงานได้พร้อมๆกัน และแน่นอนว่าต้องมีซอฟต์แวร์ที่เข้ามาทำหน้าที่นี้ ซึ่งก็คือไดรเวอร์ USB นั่นเอง

ไดรเวอร์อุปกรณ์ USB ของอุปกรณ์แต่ละตัวจะส่งการร้องรอเพื่อการติดต่อ (request) ลงมายังไดรเวอร์ USB และเมื่อไดรเวอร์ USB รับทราบความต้องการการติดต่อของอุปกรณ์ครบทุกๆ ตัวที่

เชื่อมต่ออยู่กับบัสแล้ว ก็จะพิจารณาว่า ในรอบการรับส่งข้อมูลหนึ่งๆนั้นอุปกรณ์แต่ละตัวสามารถรับส่งข้อมูลได้มากเท่าใด หากปริมาณข้อมูลที่ต้องการรับส่งมีมากก็จะตัดแบ่งออกเป็นส่วน ๆ แล้วเก็บไว้เพื่อรอส่งในรอบถัดไป โดยปริมาณข้อมูลที่ส่งได้ของอุปกรณ์แต่ละตัวจะถูกพิจารณาจากจากชนิดของการถ่ายทอดข้อมูล (transfer type) ว่า อุปกรณ์ใดใช้การถ่ายทอดข้อมูลแบบใดและการรับส่งข้อมูลชนิดนั้นมีลำดับความสำคัญมากน้อยเพียงใด

2.3.3 ไดรเวอร์โฮสคอนโทรลเลอร์

หลังจากไดรเวอร์USB พิจารณาแล้วว่าอุปกรณ์แต่ละตัวส่งข้อมูลได้เท่าใดบ้าง มันจะส่งข้อมูลของอุปกรณ์แต่ละตัวที่จะติดต่อในรอบการติดต่อนั้นๆมายังไดรเวอร์โฮสคอนโทรลเลอร์จากนั้นไดรเวอร์โฮสคอนโทรลเลอร์จะจัดเรียงลำดับข้อมูลของอุปกรณ์แต่ละชนิดลงเป็นเฟรมข้อมูล เพิ่มเติมส่วนประกอบต่างๆของเฟรมข้อมูลให้ครบตามมาตรฐานการถ่ายทอดข้อมูลแบบUSB แล้วส่งข้อมูลทั้งหมดไปยังฮาร์ดแวร์ USB โฮสคอนโทรลเลอร์เพื่อส่งข้อมูลทั้งหมดออกไปยังอุปกรณ์ต่างๆ ลำดับและขั้นตอนการทำงานของซอฟต์แวร์ควบคุมการทำงานของพอร์ตUSB

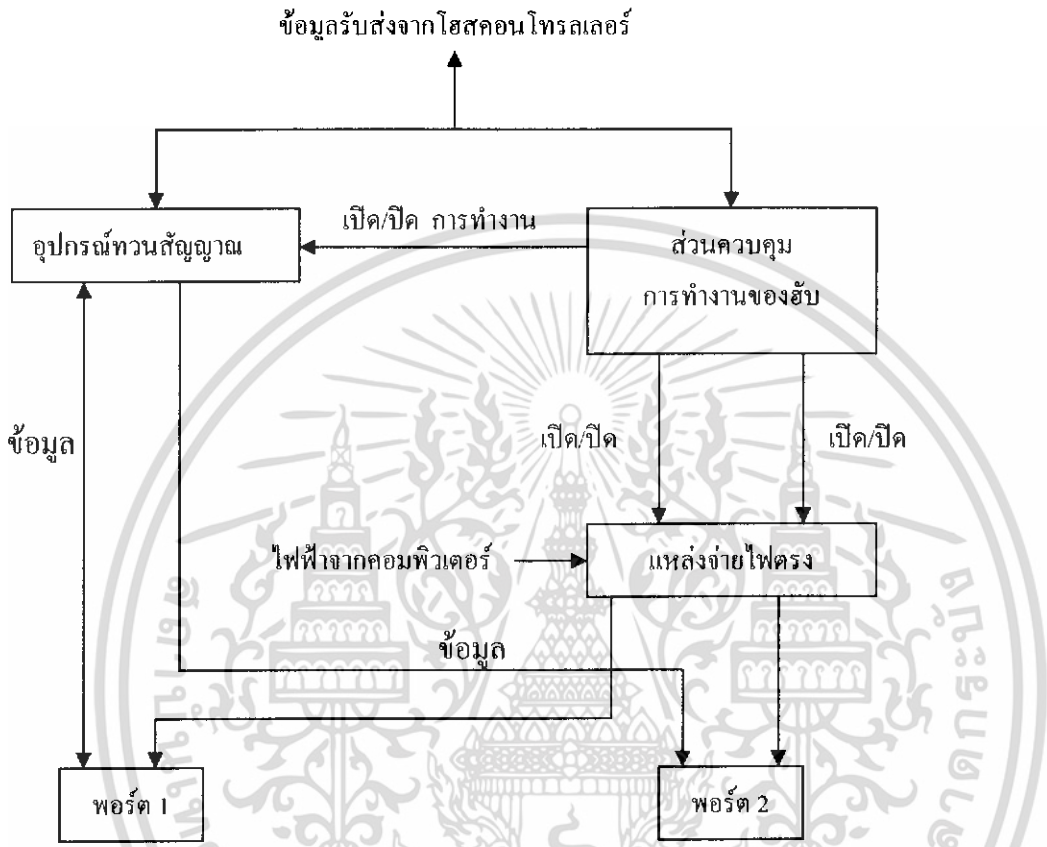
2.4 ส่วนประกอบทางฮาร์ดแวร์

จากส่วนประกอบ3ส่วนที่ผ่านมาเป็นส่วนประกอบด้านซอฟต์แวร์ ซึ่งจะคอยควบคุมจัดการการทำงานของอุปกรณ์ที่มาต่อร่วมกันทั้งหมดให้เป็นไปอย่างราบรื่น ในส่วนถัดไปจะเป็นหน้าที่ของส่วนประกอบทางฮาร์ดแวร์ที่ทำหน้าที่รับส่งสัญญาณกับตัวอุปกรณ์ต่างๆโดยส่วนประกอบตัวแรกที่จะกล่าวถึงคือ USB โฮสคอนโทรลเลอร์ (USB host controller) / รูตฮับ (root hub)

2.4.1 USB โฮสคอนโทรลเลอร์ /USB รูตฮับ

USB โฮสคอนโทรลเลอร์ มีหน้าที่สร้างสัญญาณข้อมูลทางไฟฟ้าแล้วส่งต่อไปยัง รูตฮับเพื่อกระจายออกไปยังอุปกรณ์ต่างๆโดยมันจะสร้างสัญญาณข้อมูลการติดต่อรูปแบต่างๆตามที่ไดรเวอร์โฮสคอนโทรลเลอร์กำหนดมาให้จากนั้นแปลงข้อมูลที่จะส่งมาแบบขนานเป็นแบบอนุกรมเพื่อใช้ในการส่งต่อไป เมื่อแปลงสัญญาณที่ต้องการส่งมาถึงรูตฮับ รูตฮับจะส่งสัญญาณนั้นออกไปยังบัสเพื่อส่งต่อไปยังอุปกรณ์ต่างๆ นอกจากนั้นรูตฮับยังมีหน้าที่สำคัญอีก 4 อย่างคือ

1. ควบคุมการใช้พลังงานของอุปกรณ์ที่มาต่อ
2. ตรวจสอบการเชื่อมต่อของอุปกรณ์ว่ามีอุปกรณ์ต่ออยู่หรือไม่
3. เปิดหรือเอ็นเอเบิลการใช้พอร์ตเมื่อมีอุปกรณ์ต่ออยู่ และปิดหรือดิสเอเบิลการใช้งานเมื่อปลด อุปกรณ์ออกไปแล้ว
4. รายงานสถานะของแต่ละพอร์ตเมื่อไดรเวอร์โฮสคอนโทรลเลอร์ร้องขอมา



รูปที่ 2.3 ไดอะแกรมการทำงานของUSB รูตฮับอย่างง่าย

2.4.2 USB ฮับ

หน้าที่หลักของUSBฮับ คือ ขยายการเชื่อมต่อให้อุปกรณ์จำนวนมากๆสามารถเชื่อมต่อเข้ากับระบบบัสได้ โดยการทำงานหลักของUSBฮับนั้นมีอยู่ 2 ส่วนคือ ทำหน้าที่เป็นตัวทวนสัญญาณ (repeater) และตัวจัดการพลังงาน (power management)

ในส่วนของการทวนสัญญาณUSBฮับจะต้องรับสัญญาณจากโฮสมา แล้วส่งกระจายออกไปยังพอร์ตทุกๆพอร์ตและรับสัญญาณจากแต่ละพอร์ต แล้วจับมารวมกันเพื่อส่งกลับไปที่โฮส สำหรับส่วนของการจัดการพลังงานนั้นมีหน้าที่เหมือนกับรูตฮับก็คือ ตรวจสอบว่ามีการต่ออยู่ของอุปกรณ์ที่พอร์ตใดบ้าง หากมีอุปกรณ์ต่ออยู่ก็เปิดการใช้งานพอร์ตนั้นๆหากไม่มีอุปกรณ์ ต่ออยู่ก็ปิดการใช้งาน ตรวจสอบการเชื่อมต่อหรือปลดออกของอุปกรณ์เพื่อรายงานผลเมื่อโฮสคอนโทรลเลอร์ร้องขอ และป้องกันอุปกรณ์ที่ต่ออยู่ในแต่ละพอร์ตเพื่อไม่ให้ดึงกระแสไฟฟ้าเกินกว่าที่กำหนด

2.4.3 อุปกรณ์USB

ส่วนประกอบทางฮาร์ดแวร์ส่วนสุดท้ายที่ต้องรู้จักคือ อุปกรณ์USB ซึ่งก็คือ อุปกรณ์ต่างๆที่เชื่อมต่อกับคอมพิวเตอร์ด้วยพอร์ตUSBสามารถแบ่งเป็น2ชนิดในการถ่ายทอดข้อมูลคือ

1.อุปกรณ์ความเร็วต่ำ (low-speed devices) ถ่ายทอดข้อมูลด้วยความเร็ว 1.5 เมกะบิตต่อวินาที (Mb/sec)

2.อุปกรณ์ความเร็วเต็มที่ (full speed devices) รับส่งข้อมูลด้วยความเร็ว 12 เมกะบิตต่อวินาที (Mb/sec)

ปัจจุบันนี้อุปกรณ์ที่มีจำหน่ายอยู่ตามท้องตลาดมีอยู่เป็นจำนวนมาก อาทิ คีย์บอร์ด, เมาส์, จอยสติ๊ก เหล่านี้คืออุปกรณ์ USB ความเร็วต่ำ ส่วนจอมอนิเตอร์, ลำโพง, เครื่องพิมพ์, กล้องถ่ายรูปดิจิตอล, ซีดีรอมไดรฟ์, เครื่องเล่นMP3 จัดเป็นอุปกรณ์ USB ความเร็วสูง อุปกรณ์บางตัวจะบรรจุความสามารถของ USB ฮับ เข้าไปด้วย ทำให้สามารถนำอุปกรณ์อื่นๆ มาเชื่อมต่อได้ เหมือนกับการต่อเข้ากับฮับ อุปกรณ์ลักษณะนี้เรียกว่า Compound USB Device ตัวอย่างของอุปกรณ์ที่มีฮับ อยู่ภายในได้แก่ จอมอนิเตอร์หรือเครื่องพิมพ์ เป็นต้น

นอกจากการแบ่งชนิดของอุปกรณ์ตามความเร็วในการถ่ายทอดข้อมูลแล้ว อาจจะแบ่งกลุ่มตามการใช้พลังงานของตัวอุปกรณ์เองก็ได้ ซึ่งสามารถแบ่งได้อีก 2 ชนิดคือ

1.อุปกรณ์ที่ใช้ไฟเลี้ยงจากบัส (bus powered device) คืออุปกรณ์ที่ใช้ไฟเลี้ยงจากบัสโดยตรง ไม่ต้องมีแหล่งจ่ายไฟภายนอกเพิ่มเติม

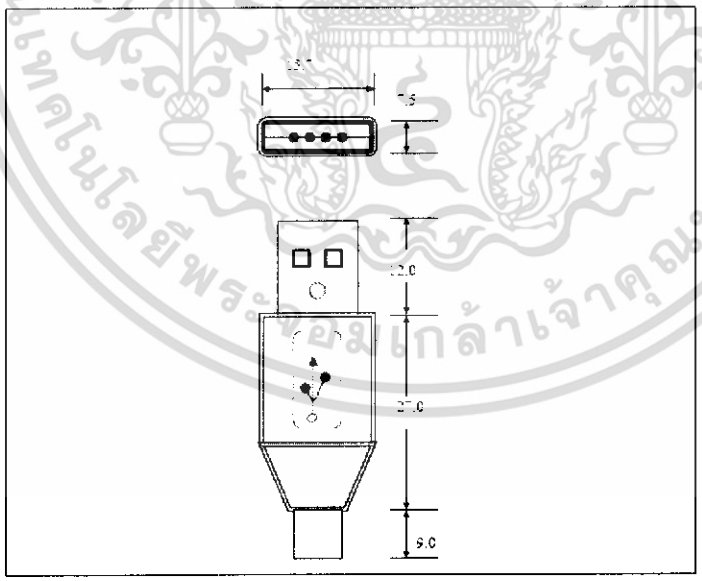
2.อุปกรณ์ที่ใช้ไฟเลี้ยงจากตัวเอง (self powered device) คือ อุปกรณ์ที่มีแหล่งจ่ายไฟในตัวไม่ต้องอาศัยไฟเลี้ยงจากบัส

2.5 สายเชื่อมต่อและคอนเน็กเตอร์ของพอร์ต USB

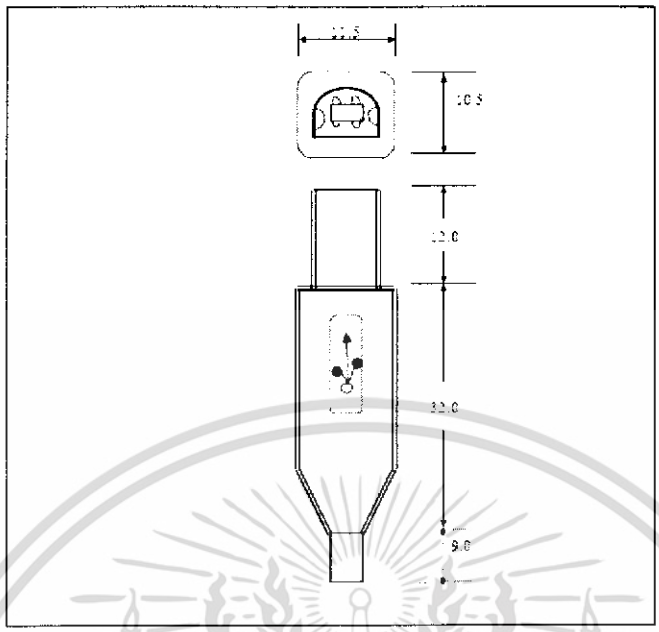
พอร์ตUSBถูกออกแบบมาสำหรับการเชื่อมต่อระหว่างตัวอุปกรณ์ต่างๆกับคอมพิวเตอร์ซึ่งอาศัยสายนำสัญญาณเป็นตัวกลาง และเนื่องจากความเร็วในการถ่ายทอดข้อมูลที่สูงประกอบกับความสามารถที่จะต่อหรืออุปกรณ์ต่างๆ โดยไม่ต้องปิดเครื่องคอมพิวเตอร์ทำให้ต้องมีการกำหนดคุณสมบัติของสายนำสัญญาณและคอนเน็กเตอร์เชื่อมต่อ เพื่อให้สามารถรองรับการทำงานดังกล่าวได้

2.6 คอนเน็กเตอร์ USB

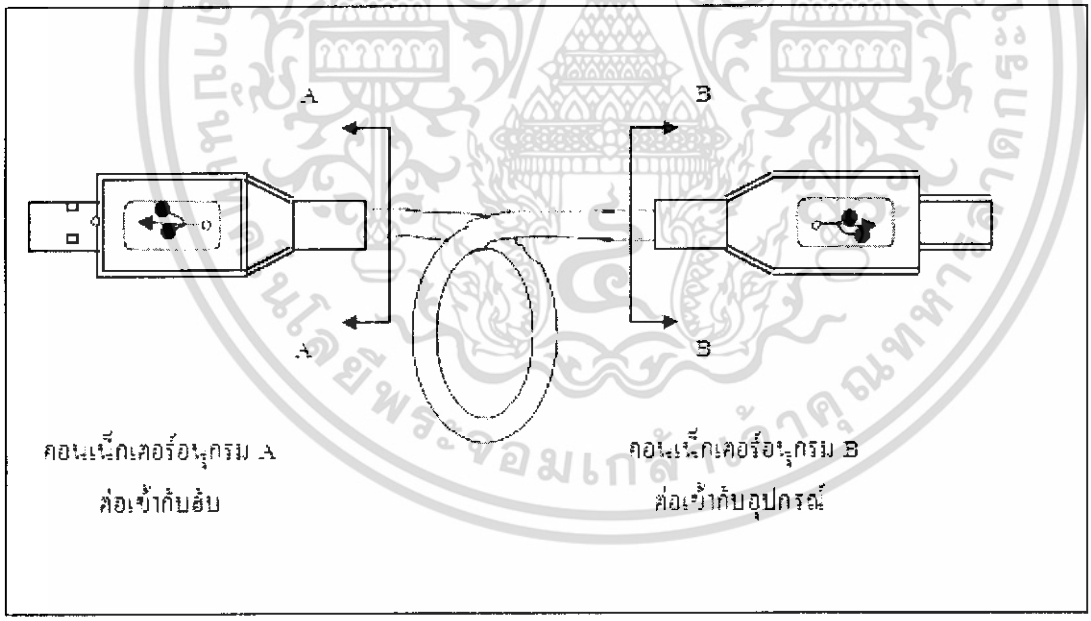
สายเชื่อมต่อUSBถูกออกแบบมาสำหรับการเชื่อมต่อระหว่างอุปกรณ์ USB กับฮับ ซึ่งอาจจะเป็นรูทฮับซึ่งอยู่หลังเครื่องคอมพิวเตอร์ ฮับที่รวมอยู่ในตัวอุปกรณ์(compound device)หรือฮับที่เป็นตัวเดี่ยวๆ (USB hub) อุปกรณ์หลายชนิดมีมีสายต่อด้านอุปกรณ์ไว้อย่างถาวรด้านหนึ่งและมีคอนเน็กเตอร์ตัวผู้ไว้สำหรับต่อเข้ากับฮับอีกด้านหนึ่ง แต่มีอุปกรณ์หลายชนิดที่ไม่มีการต่อถาวรไว้ โดยที่ตัวอุปกรณ์จะมีคอนเน็กเตอร์ USB ตัวเมียไว้เพื่อให้ใช้งานกับสายUSBมีหัวคอนเน็กเตอร์ทั้งสองด้าน จากอุปกรณ์ที่ไม่มีสายถาวรต่อไว้จะเห็นได้ว่าต้องใช้สายเชื่อมต่อซึ่งมีคอนเน็กเตอร์ ทั้งสองด้าน ซึ่งถ้าใช้คอนเน็กเตอร์แบบเดียวกันทั้งหมดอาจจะทำให้ผู้ใช้นำคอนเน็กเตอร์ด้านหนึ่งต่อเข้ากับฮับพอร์ตหนึ่งแล้วตอปลายอีกด้านต่อเข้ากับฮับอีกปลายหนึ่งได้(เพราะฮับจะมีพอร์ตอยู่หลายพอร์ต) เพื่อป้องกันเหตุการณ์เช่นนี้จึงมีการกำหนดมาตรฐานของคอนเน็กเตอร์ USB ให้มี 2 รูปแบบดังนี้



รูปที่ 2.4 รูปร่างและขนาดของคอนเน็กเตอร์ USB ในอนุกรม type A



รูปที่ 2.5 รูปร่างและขนาดของคอนเน็กเตอร์ USB ในอนุกรม type B



รูปที่ 2.6 แสดงตัวอย่างสายสัญญาณที่ใช้เชื่อมต่อพอร์ต USB

1. คอนเน็กเตอร์อนุกรม type A เป็นคอนเน็กเตอร์ด้านฮับที่เชื่อมต่อระหว่างUSBพอร์ตฮับ(ทั้งรูทฮับและUSBฮับทั่วไป) กับสายเชื่อมต่อกับอุปกรณ์ นั่นคือคอนเน็กเตอร์ตัวเมียจะติดตั้งอยู่กับฮับและคอนเน็กเตอร์ตัวผู้จะติดอยู่กับสายที่ต่อออกมาจากตัวอุปกรณ์ ดังในรูปที่ 2.1

2. คอนเน็กเตอร์อนุกรม typeB เป็นคอนเน็กเตอร์ด้านอุปกรณ์ที่เชื่อมต่อสายเข้ากับอุปกรณ์ USB นั่นคือคอนเน็กเตอร์ตัวเมียจะติดตั้งอยู่ในตัวอุปกรณ์ USB ส่วนคอนเน็กเตอร์ตัวผู้ก็จะอยู่ที่สายที่สายที่ต่อออกมาจากฮับ(ถ้าอุปกรณ์ใดที่มีสายต่อออกมาจากอุปกรณ์อย่างถาวรจะไม่มีการใช้คอนเน็กเตอร์แบบนี้) มีรูปร่างแสดงในรูปที่ 2.5 ในรูปที่ 2.6 แสดงการใช้สายUSB ที่ใช้ คอนเน็กเตอร์ทั้งสองรูปแบบ

อย่างไรก็ตามเนื่องจากในปัจจุบัน(รวมถึงแนวโน้มในอนาคต)อุปกรณ์เครื่องใช้ต่างๆมีขนาดเล็กลงมากทำให้เกิดข้อจำกัดในเรื่องขนาดคอนเน็กเตอร์บนตัวอุปกรณ์ แต่จากรูปที่3-2 จะเห็นได้ว่าขนาดของคอนเน็กเตอร์นั้นใหญ่พอสมควร ทางบริษัทที่ผลิตอุปกรณ์ต่างๆจึงมีการลดขนาดของคอนเน็กเตอร์ลงเพื่อให้ใช้งานกับอุปกรณ์ของตัวเองได้โดยยังคงรูปร่างไว้คงเดิมแต่ลดสัดส่วนความกว้าง-ยาวลง ดังจะเห็นได้จากกล้องดิจิตอล เครื่องเล่น MP3 แบบพกพา เครื่องเล่นมินิดิสก์ เป็นต้น

2.7 หน้าสัมผัสคอนเน็กเตอร์ USB

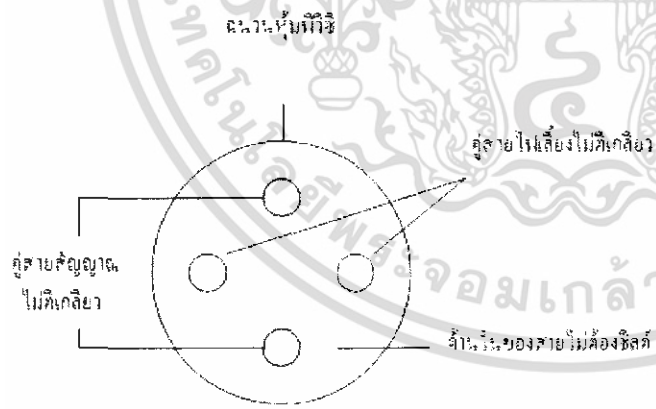
ถึงแม้มีการแบ่งคอนเน็กเตอร์ออกเป็น 2 แบบแต่ทั้งสองแบบนั้นใช้สายนำสัญญาณเหมือนกันโดย ใช้สาย 2 เส้นในการส่งไฟเลี้ยงให้กับอุปกรณ์ และสายอีก 2 เส้นสำหรับรับและส่งข้อมูล แต่เนื่องจากระบบ USB นั้นออกแบบมาเพื่อให้สามารถเชื่อมต่อหรือปลดอุปกรณ์ออกจากระบบได้ในระหว่างการใช้งาน ดังนั้นหน้าสัมผัสของคอนเน็กเตอร์จึงมีการออกแบบให้มีความสามารถพิเศษเพื่อให้รองรับคุณสมบัติดังกล่าวได้ นั่นคือหน้าสัมผัสของสาย 2 เส้นที่ใช้ ส่งไฟเลี้ยงจะยื่นออกมายาวกว่าหน้าสัมผัสที่ใช้รับส่งข้อมูล(หน้าสัมผัสสำหรับไฟเลี้ยงยาว 7.41 มิลลิเมตร หน้าสัมผัสสำหรับรับส่งข้อมูลยาว 6.41 มิลลิเมตร)การออกแบบคอนเน็กเตอร์เช่นนี้ จะทำให้ตัวอุปกรณ์ได้รับไฟเลี้ยงก่อนที่จะได้รับสัญญาณข้อมูลมาเชื่อมต่ออุปกรณ์ตัวใหม่เข้าไปในระบบ ทำให้อุปกรณ์ที่เชื่อมต่อเข้าไปใหม่สามารถทำงานได้ทันทีโดยไม่ได้รับความเสียหาย เพราะถ้าหากอุปกรณ์ได้รับสัญญาณข้อมูลก่อนที่ได้รับไฟเลี้ยงจะเกิดความเสียแก่ไอซีที่อยู่ภายในได้ ขาของคอนเน็กเตอร์ทำหน้าที่ตามที่แสดงในตารางที่ 2.1

ตารางที่ 2.1 หน้าที่ของขาคอนเน็กเตอร์แต่ละขา

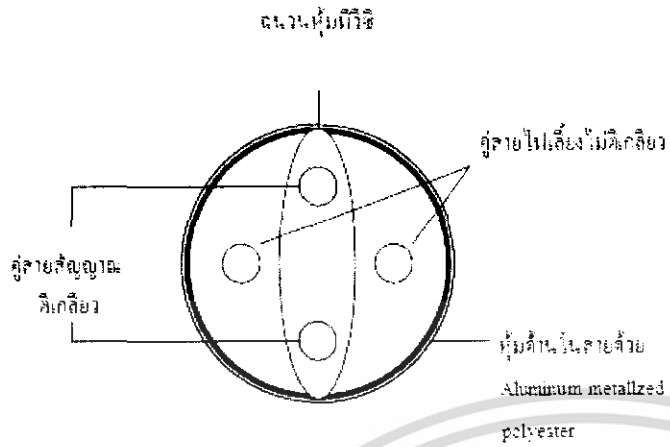
หมายเลขขาสัญญาณ	ชื่อขาสัญญาณ	สีของสายสัญญาณ
1	ไฟเลี้ยง +5V	แดง
2	D - (ข้อมูลลบ)	ขาว
3	D+ (ข้อมูลบวก)	เขียว
4	กราวด์	ดำ

2.8 สายนำสัญญาณ USB

การถ่ายทอดข้อมูลภายในสาย USB นั้นมีความเร็วสูงในระดับ 1 Mb/s ถึง 12Mb/s การถ่ายทอดข้อมูลที่มีความเร็วสูงขนาดนี้จะเกิดการแพร่กระจายสนามแม่เหล็กไฟฟ้าออกมาจากสายส่ง ทำให้ต้องมีการกำหนดคุณสมบัติของสายส่งในหลายๆด้านเพื่อไม่ให้เกิดการแพร่กระจายของสนามแม่เหล็กไฟฟ้าเหล่านี้ เนื่องจาก USB มีการแบ่งความเร็วของการถ่ายทอดข้อมูลออกเป็น 2 ระดับ คือ แบบความเร็วเต็มที่ (full speed) ที่มีอัตราเร็ว 12 Mb/s และแบบความเร็วต่ำ (low speed) ที่มีอัตราเร็ว 1.5 Mb/s ทำให้คุณสมบัติของสายนำสัญญาณของอุปกรณ์แต่ละชนิดจะถูกกำหนดแยกแตกต่างกันไป



รูปที่ 2.7 โครงสร้างความในของสาย USB แบบความเร็วต่ำ



รูปที่ 2.8 โครงสร้างภายในสาย USB แบบความเร็วเต็มที่สายนำสัญญาณความเร็วต่ำ

จากรูปที่ 2.8 แสดงให้เห็นภาพหน้าตัดของสายนำสัญญาณความเร็วต่ำ ชั้นนอกสุดของสายสัญญาณจะเป็นฉนวนหุ้มธรรมดา ภายในมีสายสัญญาณจะมีสายตัวนำ 4 เส้นโดย 2 เส้นใช้ส่งไฟเลี้ยงกำหนดให้ลวดทองแดงเบอร์ 20-28 AWG ไม่ดีเกลียว ส่วนอีก 2 เส้นใช้ถ่ายทอดข้อมูลกำหนดให้ใช้ลวดทองแดงเบอร์ 28 AWG ไม่ดีเกลียว สายส่งข้อมูลความเร็วต่ำนี้ไม่จำเป็นต้องชีลด์ ความยาวทั้งหมดของสายไม่เกิน 3 เมตร

2.9 สายนำสัญญาณความเร็วสูง

ข้อกำหนดของสายนำสัญญาณความเร็วสูงของ USB จะมีมากกว่าสายนำสัญญาณความเร็วต่ำอยู่หลายๆด้าน สาย 2 เส้นที่ใช้ถ่ายทอดข้อมูลกำหนดให้ใช้ลวดทองแดงเบอร์ 28AWG เช่นเดียวกับสายนำสัญญาณความเร็วต่ำ แต่สาย 2 เส้นนี้จะต้องดีเกลียว และชั้นนอกสุดถัดจากฉนวนจะต้องถูกหุ้มด้วยอลูมิเนียมฟอยล์ ดังในรูปที่ 2.5 ความยาวของสายสัญญาณไม่เกิน 5 เมตร จะต้องมีค่าหน่วงเวลาสัญญาณ(propagation delay)ไม่เกิน 30 นาโนวินาทีตลอดความยาวสาย 5 เมตร แต่ค่าหน่วงเวลามากกว่า 30นาโนวินาที ค่าความยาวสูงสุดของสายสัญญาณก็จะลดลงเป็นสัดส่วนในตารางที่ 2.2

ค่าอัตราการหน่วงเวลาของสาย USB	สีของสายสัญญาณ
9.0 นาโนวินาทีต่อเมตร	3.3 เมตร
8.0 นาโนวินาทีต่อเมตร	3.7 เมตร
7.0 นาโนวินาทีต่อเมตร	4.3 เมตร
6.5 นาโนวินาทีต่อเมตร	4.6 เมตร

ตารางที่ 2.2 แสดงอัตราการหน่วงเวลาของสาย USB เมื่อเทียบกับความยาวของสายสัญญาณ

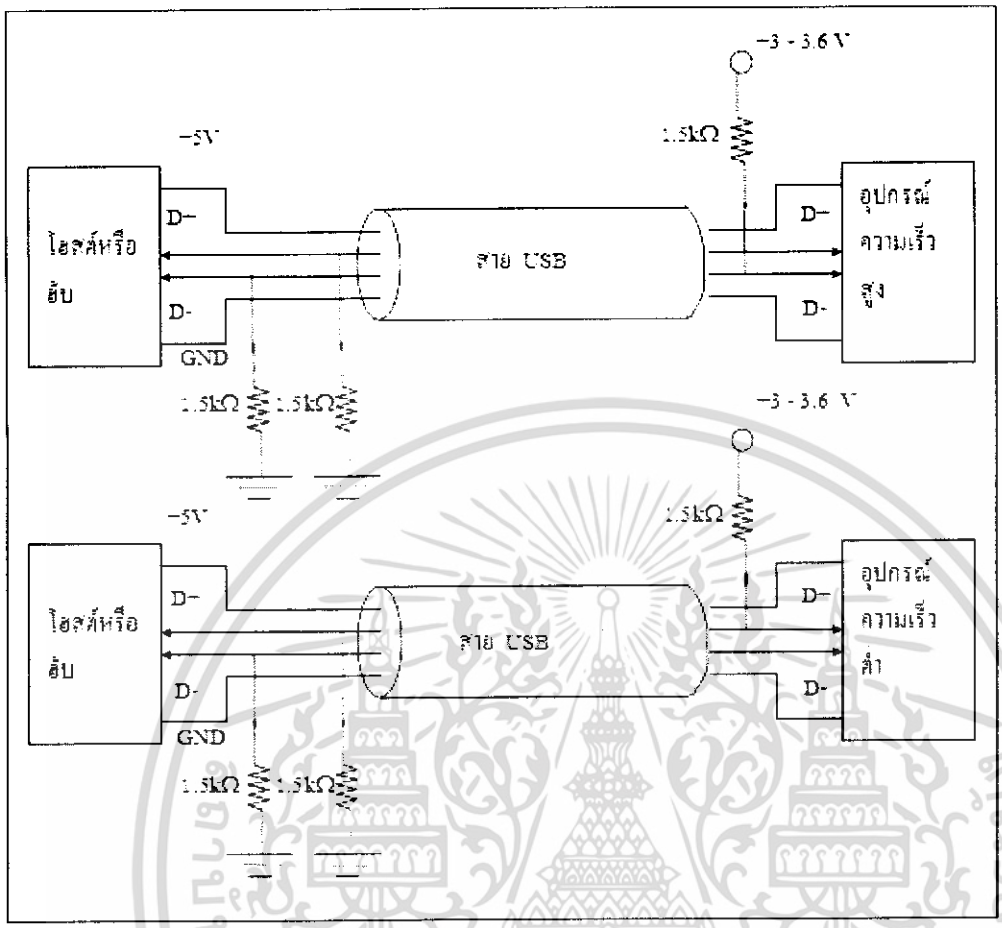
2.10 การตรวจสอบการเชื่อมต่อและความเร็วของอุปกรณ์

คุณสมบัติเด่นข้อหนึ่งของ USB คือ สามารถตรวจสอบการเชื่อมต่อของอุปกรณ์ตัวใหม่และตั้งค่าเริ่มต้นที่จำเป็นได้อัตโนมัติ แต่เนื่องจากอุปกรณ์ USB แบ่งออกเป็นอุปกรณ์ความเร็วสูงและต่ำ ถ้าส่งข้อมูลผิดความเร็วก็จะไม่สามารถสื่อสารกับอุปกรณ์นั้นๆได้ นั่นหมายความว่าจะต้องมีสัญญาณรูปแบบพิเศษที่ใช้แจ้งสถานะการเชื่อมต่อหรือปลดออกรวมถึงระบุความเร็วของอุปกรณ์นั้น ๆ ได้ด้วย

การตรวจสอบการเชื่อมต่อของ USB นั้นจะตรวจสอบจากการเปลี่ยนแปลงของระดับสัญญาณในสายสัญญาณ D - และ D + โดยสายข้อมูลทั้งสองที่ด้านฮับก็จะถูกต่อด้วยตัวต้านทาน 15 kΩ พูลดาวน์ไว้ทั้งสองเส้น (การพูลดาวน์ (pull –down) เป็นการต่อขาข้างหนึ่งของอุปกรณ์เข้ากับสายสัญญาณอีกข้างหนึ่งต่อกราวด์ของระบบ) ซึ่งจะส่งผลให้สายสัญญาณทั้งสองมีระดับแรงดันเป็น 0 Vในขณะที่ไม่มีอุปกรณ์ใด ๆ ต่ออยู่

ที่ด้านอุปกรณ์ สำหรับอุปกรณ์ความเร็วต่ำสายสัญญาณ D - จะต่อตัวต้านทาน 1.5 kΩ พูลอัปกับแรงดัน 3.0 - 3.6 V(การพูลอัป (pull-up) เป็นการต่อขาข้างหนึ่งของอุปกรณ์เข้ากับสัญญาณอีกข้างหนึ่งต่อไฟเลี้ยง ทำให้สถานะที่จุดนั้นเกิดเป็นลอจิก " 1 ") ส่วนอุปกรณ์ความเร็วสูง สายสัญญาณที่ถูกพูลอัปคือ สายสัญญาณ D + เมื่ออุปกรณ์ตัวใหม่ถูกต่อเข้ากับฮับจะเกิดการแบ่งแรงดันจากตัวต้านทาน 2 ตัวก็คือตัวต้านทาน 1.5 kΩ ที่พูลอัปทางอุปกรณ์และตัวต้านทาน15 KΩ ที่พูลดาวน์อยู่ทางฮับ ทำให้แรงดันของสายสัญญาณเพิ่มจาก 0 V ขึ้นไปที่ 90% ของไฟเลี้ยง

(คำนวณได้จาก $\frac{15\times10^{3}}{(1.5+15)\times10^{3}\times Vcc}$)



รูปที่ 2.9 การตรวจสอบการเชื่อมต่อของระบบ USB

การเปลี่ยนแรงดันนี้จะทำให้ฮับรู้ว่ามีอุปกรณ์ตัวใหม่ต่อเข้ากับระบบ และจะทำให้รู้ว่าอุปกรณ์ถูกปลดออกจากระบบเช่นเดียวกัน โดยถ้าสัญญาณที่เกิดการเปลี่ยนแปลงแรงดันนี้เป็นสัญญาณ D - หมายความว่าอุปกรณ์ที่นำมาต่อเป็นอุปกรณ์ความเร็วต่ำ แต่ถ้าสัญญาณ D + หมายความว่าอุปกรณ์ที่นำมาต่อเป็นอุปกรณ์ความเร็วสูงถูกนำมาต่อเข้ากับระบบ

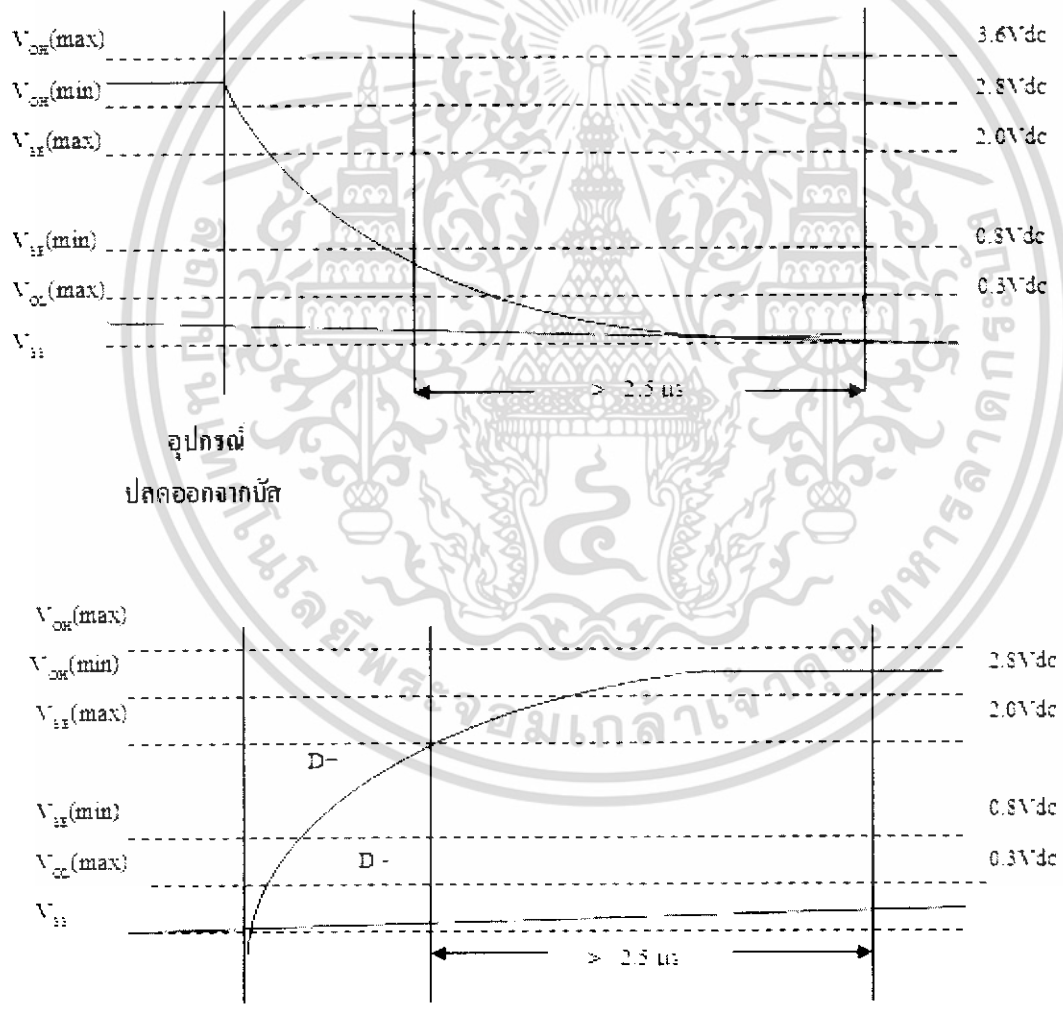
ในรูปที่ 2.9 แสดงให้เห็นถึงระดับแรงดันในสายสัญญาณเมื่อมีการเชื่อมต่ออุปกรณ์ใหม่และเมื่อปลดอุปกรณ์ออกจากบัส ถ้าแรงดันของสัญญาณ D - หรือ D + ตกลงต่ำกว่าค่า VSE (max) หรือ 2.0V เป็นเวลานานกว่า 2.5ไมโครวินาที จะถือว่ามีการต่ออุปกรณ์ตัวใหม่เข้ามาในระบบ ผลการตรวจสอบที่ได้ฮับก็จะเก็บไว้ในรีจิสเตอร์ สถานะ ซึ่งโฮสจะคอยมาอ่านเป็นระยะๆ เพื่อตรวจสอบการเชื่อมต่อของอุปกรณ์

2.11 เทคนิคการเข้ารหัสสัญญาณ

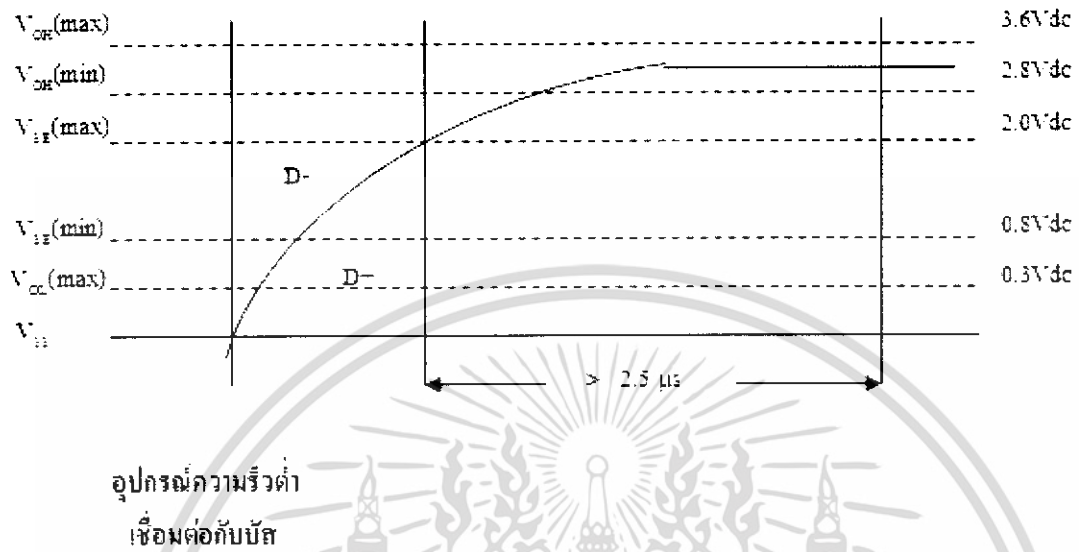
การเข้ารหัสสัญญาณข้อมูลก่อนที่จะส่งออกไปใช้เทคนิค 2 แบบประกอบด้วยกันคือ การเข้ารหัสแบบ NRZI และการเติมบิตสตัฟ (bit stuffing)

2.11.1 การเข้ารหัส NRZI

ก่อนส่งข้อมูลไปในสายข้อมูลจะถูกเข้ารหัสแบบ NRZI (Non Return to Zero Inverted) ประโยชน์การเข้ารหัสแบบนี้คือ สามารถแฝงสัญญาณนาฬิกาส่งไปกับข้อมูลได้ทำให้อุปกรณ์ปลายทางสามารถซิงโครไนซ์สัญญาณนาฬิกาของตัวเองให้ตรงกับสัญญาณนาฬิกาของโฮสได้ ทำให้สามารถอ่านข้อมูลที่ส่งมาได้อย่างถูกต้อง

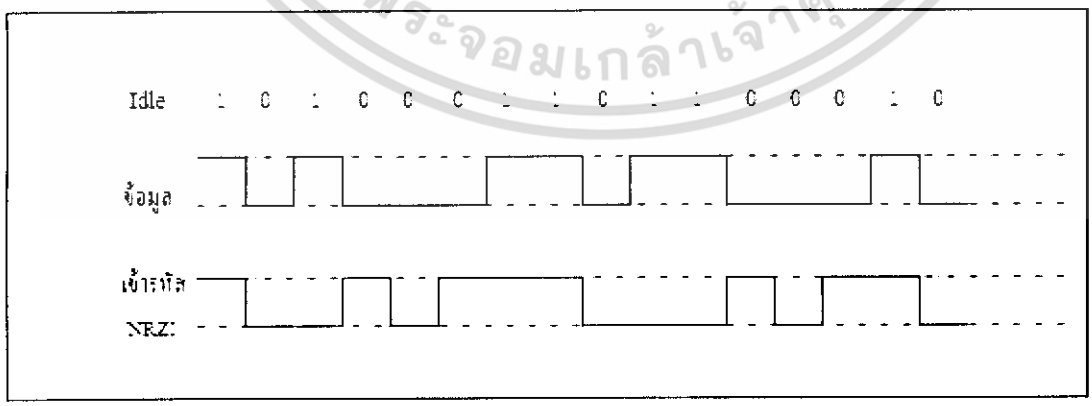


รูปที่ 2.10 ระดับสัญญาณภายในสาย D +, D - เมื่อมีการเชื่อมต่อและปลดอุปกรณ์



รูปที่ 2.10 (ต่อ) ระดับสัญญาณภายในสาย D + , D - เมื่อมีการเชื่อมต่อและปลดอุปกรณ์

ในรูปที่ 2.11 แสดงให้เห็นการเข้ารหัสแบบ NRZI ของข้อมูล การเปลี่ยนแปลงของระดับสัญญาณจากระดับสูงไปต่ำหรือจากระดับต่ำไปสูงจะถือเป็นข้อมูล " 0 " แต่ถ้าไม่เกิดการเปลี่ยนแปลงของสัญญาณจะถือเป็นข้อมูล " 1 " แต่ด้วยการเข้ารหัสวิธีนี้ หากข้อมูลที่ส่งเป็น " 1 " เรียงกันหลาย ๆ บิตก็จะไม่เกิดการเปลี่ยนแปลงระดับสัญญาณในสายสัญญาณเลย อาจทำให้เกิดความผิดพลาดในการส่งข้อมูลได้ จึงต้องมีอีกเทคนิคในการส่งเข้ามาช่วยป้องกันก็คือการเติมบิตสตัฟ

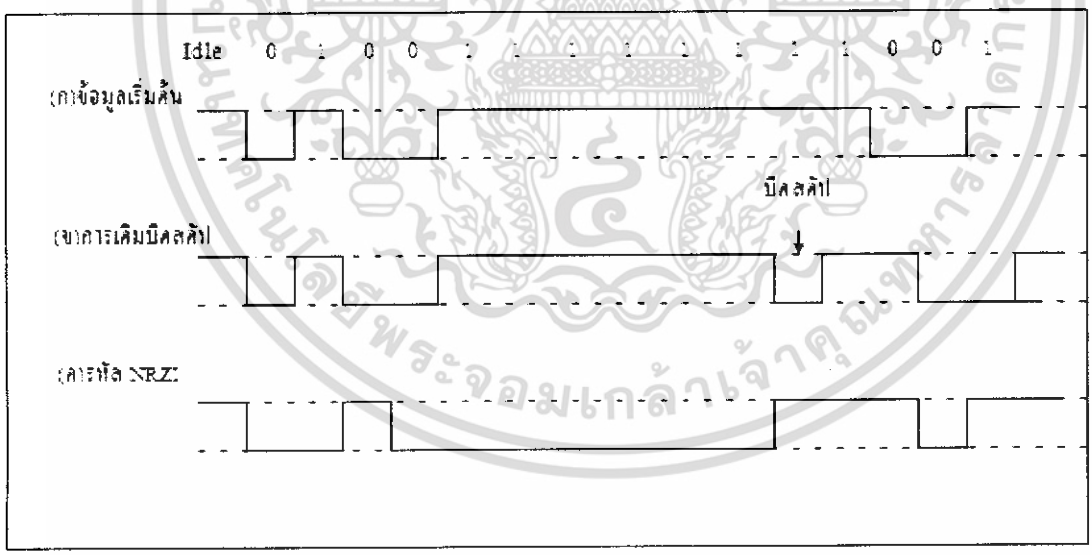


รูปที่ 2.11 การเข้ารหัส NRZI

2.11.2 การเติมบิตสตัป(Bit Stuffing)

การเติมบิตสตัฟจะเกิดขึ้นเมื่อข้อมูลที่จะส่งเป็น " 1 " ต่อเนื่องกัน 6 บิต หลังจากข้อมูล "1"เรียงกันครบ 6ตัวจะเกิดการเติม "0"ลงในข้อมูลที่จะส่งโดยอัตโนมัติ ซึ่งบิตข้อมูล " 0 " นี้เรียกว่า "บิตสตัฟ" ถึงแม้ว่าข้อมูลที่ 7 ถัดไปจะเป็น " 0 " ก็ยังคงเกิดการเติมการเติมบิตสตัฟนี้ด้วย การกระทำเช่นนี้ทำให้เกิดการส่งข้อมูลที่เข้ารหัสแบบ NRZI เกิดการเปลี่ยนแปลงอย่างน้อยทุก ๆ 7 บิตข้อมูล เมื่อฝั่งรับได้รับข้อมูล " 1 " เรียงกัน 6 บิตก็จะตัดข้อมูล " 0 " ที่ตามมาทิ้งไปทำให้ข้อมูลที่ได้รับมาถูกต้อง

จากรูปที่ 2.12 เป็นตัวอย่างของการเติมบิตสตัฟลงในข้อมูล โดยแถวบนสุดของรูปแสดงให้เห็นถึงข้อมูลที่จะส่งจริง ๆ แต่เนื่องจากข้อมูลที่จะส่งเป็นข้อมูล " 1 " ติดกัน 8 ตัวดังนั้นหลังจากการส่งข้อมูล " 1 " ที่ติดกัน 6 บิตเสร็จแล้วจะมีการเติมข้อมูล " 0 " ลงในข้อมูลหลัก เพื่อให้ เกิดการเปลี่ยนแปลงของระดับสัญญาณของการเข้ารหัสแบบ NRZI และหลังจากการเติมบิตสตัฟแล้วก็จะส่งข้อมูลหลักที่อยู่ต่อหลังไปเรื่อย ๆ จนครบ ในขณะที่ฝั่งส่งมีการเติมบิตสตัฟ ทางฝั่งรับก็จะมีการตัดบิตสตัฟทิ้งเพื่อให้ได้รับข้อมูลที่ถูกต้อง โดยการตัดก็จะใช้วิธีเดียวกับการแทรกคือตรวจสอบเมื่อพบข้อมูล " 1 " เรียงต่อเนื่องกันครบ 6 ตัว จะตัดข้อมูล " 0 " ที่ตามหลังมาทิ้งโดยอัตโนมัติ แต่หากพบว่าหลังข้อมูล " 1 " ติดกัน 6 บิตไม่เป็น " 0 " ก็แสดงว่าเกิดความผิดพลาดในการส่งข้อมูลเกิดขึ้น

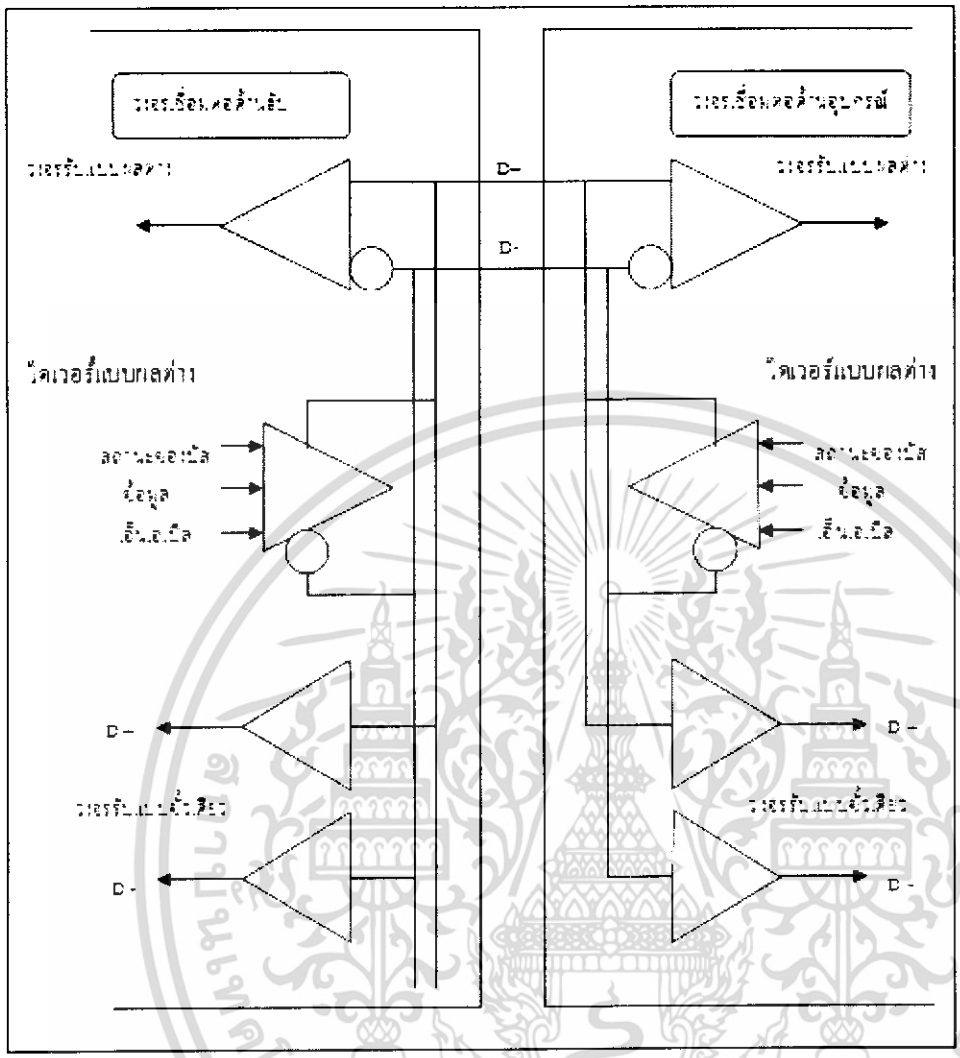


รูปที่ 2.12 การเติมบิตสตัฟ

2.12 การส่งข้อมูลในสายสัญญาณ

หลังจากข้อมูลถูกเข้ารหัสเรียบร้อยแล้ว ข้อมูลเหล่านั้นก็จะส่งออกไปยังอุปกรณ์ต่างๆที่ต่ออยู่กับฮับ แต่เนื่องจาก USB ส่งข้อมูลด้วยความเร็วที่สูงมาก โดยใช้สายสัญญาณที่ยาว (เมื่อเทียบกับสายสัญญาณข้อมูลภายในคอมพิวเตอร์)ทำให้ไม่สามารถใช้การส่งแบบขั้วเดียว (single end)ได้เพราะจะทำให้เกิดการแพร่กระจายของคลื่นแม่เหล็กไฟฟ้าไปกวนอุปกรณ์อื่นๆและยังทำให้สัญญาณรบกวนจากภายนอกเข้าไปกวนข้อมูลที่ต้องการส่งในสายสัญญาณได้ง่าย

การส่งสัญญาณที่แก้ไขปัญหา 2 ข้อที่กล่าวมาก็คือการส่งสัญญาณผลต่าง(differential pair signaling)เนื่องจากการส่งสัญญาณด้วยวิธีนี้จะทำให้สนามแม่เหล็กไฟฟ้าที่เกิดขึ้นจากสายสัญญาณทั้งสองเส้นหักล้างกัน เนื่องจากมีศักย์ตรงข้ามกัน จึงไม่เกิดการแพร่กระจายไปรบกวนอุปกรณ์อื่นๆ เมื่อมีสัญญาณรบกวนจากภายนอกเข้ามาในสาย ระดับของสัญญาณรบกวนจะเท่ากัน เพราะสายสัญญาณ 2 เส้นอยู่คู่กัน เมื่อสัญญาณรบกวนเหนี่ยวนำเข้าสู่สายเส้นหนึ่งก็จะเหนี่ยวนำเข้าสู่สายอีกเส้นในขนาดที่เท่ากัน เมื่อถึงปลายทางในวงจรภาครับจะใช้วงจรขยายผลต่าง(differential amplifier)เป็นตัวรับสัญญาณ ซึ่งวงจรนี้มีคุณสมบัติขยายสัญญาณที่มีขนาดต่างกันและลดทอนสัญญาณที่มีขนาดเท่ากัน ดังนั้นสัญญาณข้อมูลซึ่งมีศักย์ตรงข้ามกันจะถูกขยายเพื่อส่งต่อไปยังส่วนถัดไป แต่สัญญาณรบกวนซึ่งมีขนาดเท่ากันในสายทั้ง 2 เส้น จะถูกลดทอนหรือตัดทิ้งไป



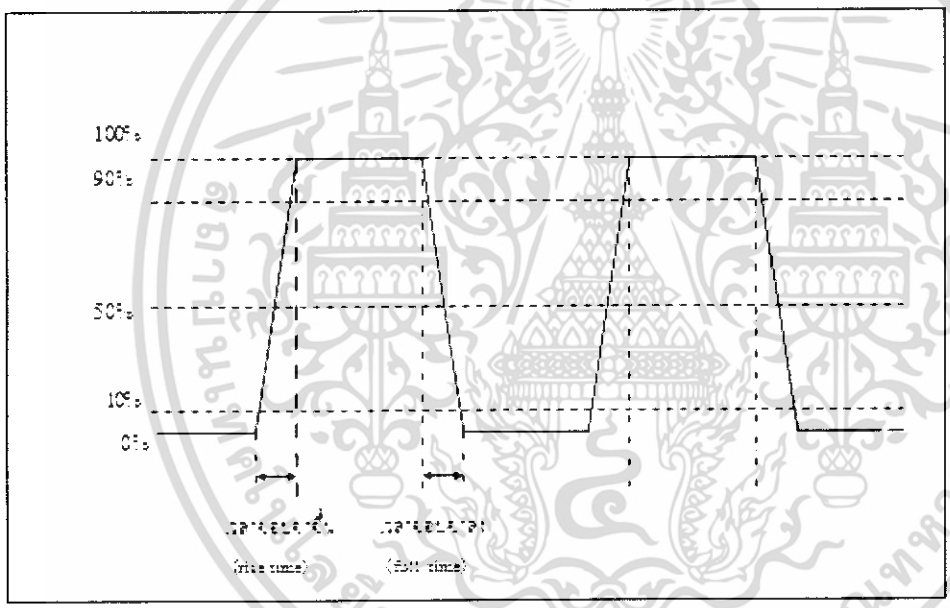
รูปที่ 2.13 วงจรเชื่อมต่อระหว่างภาคส่ง (ฮับ) และภาครับ (ตัวอุปกรณ์)

จากรูปที่ 2.13 แสดงให้เห็นวงจรภาคส่งและภาครับของฮับและอุปกรณ์ จะเห็นได้ว่าทั้งสองด้านมีวงจรที่เหมือนกันเนื่องจาก USB เป็นบัสที่รับส่งข้อมูลแบบฮาร์ฟดูเพล็กซ์ นั่นคือ ทั้งสองด้านสามรถรับส่งข้อมูลได้ทั้งคู่ แต่คนละช่วงเวลากัน ทำให้ทั้งสองด้านมีชุดวงจรที่เหมือนกัน และเนื่องจากการรับส่งข้อมูลแบบนี้มีอุปกรณ์เพียงตัวเดียวที่สามารถรับส่งข้อมูลได้ในช่วงเวลาหนึ่งๆ อุปกรณ์ที่เหลือที่ไม่ได้ใช้งาน บัสจำเป็นจะต้องปล่อยบัสสัญญาณให้มีสภาพ อิมพิแดนซ์สูง (high impedance) ดังนั้นวงจรภาคส่งของอุปกรณ์แต่ละตัวจะต้องสามารถกำหนดสถานะของเอาต์พุตให้เป็นสภาพอิมพีแดนซ์สูงได้

ส่วนวงจรขับแบบผลต่าง (differential driver) ทำหน้าที่สร้างสัญญาณข้อมูลแบบผลต่างส่งไปในสายสัญญาณ D + และ D - โดยสัญญาณในสายทั้ง 2 เส้นจะมีเฟสต่างกัน 180 องศาวงจรขับแบบผลต่างที่อยู่ในฮับ จะต้องรองรับการถ่ายทอดข้อมูลได้ทั้งแบบความเร็วสูงและต่ำ เพื่อให้สามารถ

ถ่ายทอดข้อมูลไปยังอุปกรณ์ได้ทั้งสองชนิด แต่วงจรฝั่งตัวอุปกรณ์นั้นรองรับการส่งได้เพียงชนิดเดียวตามชนิดของอุปกรณ์นั้น ๆ ก็พอเพียงพอ

คุณสมบัติที่สำคัญอีกประการหนึ่งที่จำเป็นในการออกแบบวงจรขับคือ ค่าเวลาขอบขาขึ้นและลงของสัญญาณ หรือ rise time และ fall time ของสัญญาณข้อมูล โดยเวลาขอบขาขึ้นหรือ rise time คือ เวลาที่ใช้ในการเปลี่ยนระดับแรงดันจาก 10% ไปจนถึง 90% ของแรงดันสูงสุด ในขณะที่เวลาขอบขาลงหรือ fall time คือเวลาที่ใช้ในการเปลี่ยนระดับแรงดันจาก 90% ถึง 10%ของแรงดันสูงสุด ดูรูปที่ 2.14 ประกอบค่าเวลาขอบขาขึ้นและลงของการถ่ายทอดข้อมูลความเร็วสูงจะต้องอยู่ในช่วง 4 ถึง 20 นาโนวินาที เพื่อไม่ให้เกิดผลกระทบต่อบิตข้อมูล ส่วนการถ่ายทอดข้อมูลที่ความเร็วต่ำ ค่าเวลาขอบขาขึ้นและลงจะอยู่ในช่วง 75 ถึง 300 นาโนวินาทีเพื่อให้เป็นไปตามมาตรฐาน FCC Class B



รูปที่ 2.14 แสดงช่วงเวลาขอบขาขึ้น(rise time) และขอบขาลง (fall time) ของสัญญาณ

ค่าข้อมูล " 1 " และ " 0 " ของการส่งแบบผลต่างนั้นจะดูจากความแตกต่างของระดับแรงดันในสายสัญญาณทั้ง 2 เส้น เช่น ถ้าแรงดันในสาย D + มากกว่า D - จะถือเป็นข้อมูล " 1 " แต่ถ้าแรงดันในสาย D - มากกว่า D+ จะถือเป็นข้อมูล " 0 " จากตัวอย่างจะเห็นได้ว่ามีการตีความหมายของข้อมูลได้ 2 กรณี การถ่ายทอดข้อมูลแบบความเร็วสูงและต่ำจะตีความหมายสัญญาณ 2 แบบนี้ต่างกันคือ ในโหมดความเร็วสูงข้อมูล " 1 " จะแทนด้วยระดับแรงดันในสาย D + มากกว่า D - และข้อมูล " 0 " จะกลับตรงข้ามกัน ส่วนในโหมดความเร็วต่ำ ข้อมูล " 1 " จะแทนด้วยแรงดันในสาย D- มากกว่า D + และข้อมูล " 0 " จะกลับตรงข้ามเช่นเดียวกัน หรือกล่าวง่าย ๆได้ว่าการถ่ายทอดข้อมูลในโหมดความเร็วสูงและความเร็วต่ำมีขั้วตรงข้ามกัน

ส่วนวงจรรับผลต่าง (differential receiver) ทำหน้าที่รับและขยายสัญญาณข้อมูลที่มีเฟสตรงข้าม และตัดสัญญาณที่รบกวนที่มีเฟสเหมือนกันทิ้งไป ส่วนวงจรรับแบบขั้วเดียว (single-ended receiver) ทั้ง 2 ชุดนั้นใช้สำหรับตรวจสอบสถานะของบัส เพราะว่า สถานะต่างๆของบัสจะต้องพิจารณาจากระดับแรงดันของสายทั้ง 2 เส้นแยกจากกัน เช่นการเชื่อมต่อของอุปกรณ์ตัวใหม่จะเกิดการเปลี่ยนแปลงของสัญญาณเพียงเส้นเดียว ซึ่งวงจรรับแบบผลต่างไม่สามารถตรวจสอบการเปลี่ยนของสัญญาณในสายเพียงเส้นเดียวนี้ได้ เนื่องจากวงจรสามารถตรวจสอบนี้ว่า เกิดการเปลี่ยนแปลงของสัญญาณ แต่ไม่ทราบว่าเกิดการเปลี่ยนแปลงในสายเส้นใด ตารางที่ 2.3 แสดงให้เห็นสถานะต่างๆของบัส USB

สถานะของบัส	รายละเอียด
สถานะเตรียมพร้อม(Idle) กรณีความเร็วต่ำ (LS) กรณีความเร็วเต็มที่ (FS)	ระดับแรงดันที่ขา D – มากกว่า D + เนื่องจากมีความต้านทานพูลฮัปที่ขา D – ระดับแรงดันที่ขา D + มากกว่า D - เนื่องจากมีความต้านทานพูลฮัปที่ขา D +
สถานะติดต่อซ้ำ (resume)	ข้อมูลในสายสัญญาณมีสถานะตรงข้ามกับสถานะเตรียมพร้อม
เริ่มส่งข้อมูล (SOP: Start Of Packet)	ข้อมูลในสายสัญญาณมีสถานะตรงข้ามกับสถานะเตรียมพร้อม
สิ้นสุดข้อมูล (SOP: Start Of Packet)	แรงดันที่ขา D + และ D - ต่ำกว่า 0.8 Vเป็นเวลา 2 บิตข้อมูลแล้วตามหลังด้วยสถานะเตรียมพร้อมหรือ Idle เป็นเวลา 1 บิตข้อมูล
การเชื่อมต่ออุปกรณ์	แรงดันที่ขา D + หรือ D – (ขึ้นอยู่กับชนิดของอุปกรณ์) สูงกว่า 2 V นานกว่า 2.5 ไมโครวินาที
การปลดออกของอุปกรณ์	แรงดันที่ขา D + หรือ D – (ขึ้นอยู่กับชนิดของอุปกรณ์) ต่ำกว่า 0.8 V นานกว่า 2.5 ไมโครวินาที
รีเซ็ต	แรงดันที่ขา D + หรือ D – ต่ำกว่า 0.8 V เป็นเวลานาน 10 มิลลิวินาที

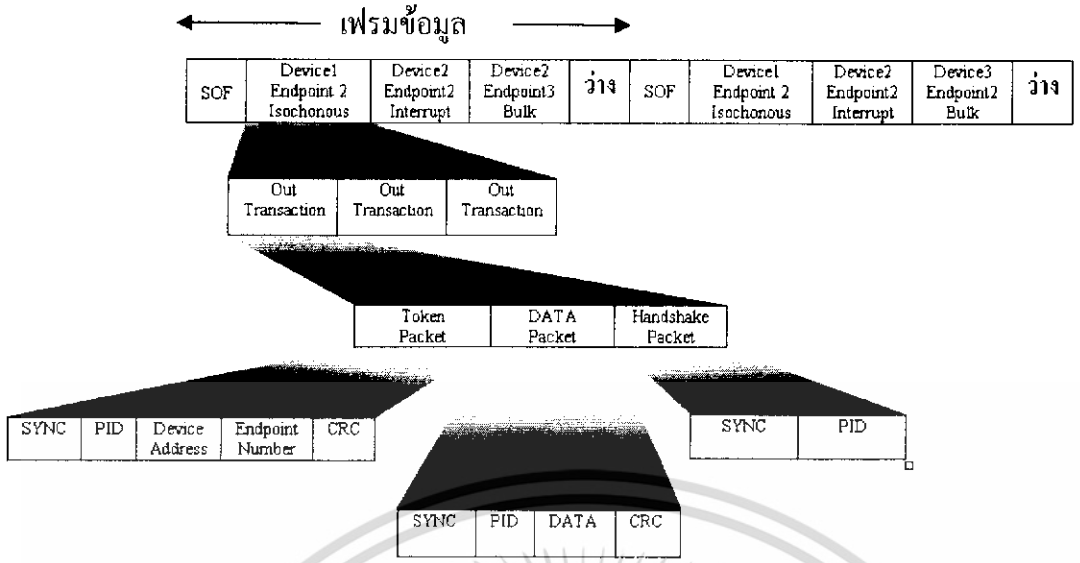
ตารางที่ 2.3 สถานะของบัส USB

บทที่ 3
การส่งถ่ายข้อมูลบนระบบบัส ยูเอสบี

การสื่อสารข้อมูลในระบบบัสยูเอสบี เป็นการสื่อสารที่มีโฮสเป็นจุดศูนย์กลาง ซึ่งโฮสมีหน้าที่ในการทำให้การส่งถ่ายข้อมูลทั้งหมดที่อยู่บนบัสเกิดขึ้นเร็วที่สุดเท่าที่จะเป็นไปได้ โดยมันจัดการขนส่งข้อมูลทีอยู่ภายในระบบบัสด้วยการแบ่งเวลาออกเป็นส่วนเล็กๆ เรียนกว่าเฟรมและสำหรับโหมดไฮสปีดจะแบ่งเป็นไมโครเฟรม โฮสจะแบ่งส่วนของเฟรมหรือไมโครเฟรมให้แก่การส่งถ่ายข้อมูลแต่ละชุด (รูปที่ 3.1) โดยช่วงเวลาของเฟรมสำหรับข้อมูลโลว์และฟูลสปีดแต่ละเฟรมจะมีค่าเท่ากับ 1 มิลลิวินาที สำหรับการส่งถ่ายข้อมูลแบบไฮสปีดจะแบ่งเฟรมแต่ละเฟรมออกเป็น 8 ส่วน โดยให้ชื่อแต่ละส่วนว่าไมโครเฟรมและมีช่วงเวลาเท่ากับ 125 มิลลิวินาที แต่ละเฟรมหรือไมโครเฟรมจะเริ่มต้นด้วยตัวเริ่มเฟรมหรือเรียกสั้นๆ ว่า SOF (Start-of-frame)

การส่งถายข้อมูลแต่ละชนิดจะประกอบไปด้วยการสื่อสารหรือเรียกทับศัพท์ว่า ทรานแซคชัน (Transaction) มากกว่า 1 ชุด สำหรับการส่งถายข้อมูลแบบคอนโทรลจะมีทรานแซคชันมากกว่า 1 ชุดเสมอเนื่องจากพวกมันจะมีขั้นตอนหรือสเตจ (Stage) มากกว่า 1 สเตจ ซึ่งแต่ละสเตจอาจจะประกอบด้วยทรานแซคชันมากกว่า 1 ชุด ส่วนการส่งถ่ายข้อมูลแบบอื่น ๆ นั้นจะใช้ทรานแซคชันหลายชุดก็ต่อเมื่อมันมีข้อมูลมากเกินกว่าที่จะใส่ลงในทรานแซคชันเดียวได้ ทรานแซคชันของการส่งถ่ายข้อมูลอาจจะทำได้สำเร็จภายในหนึ่งเฟรมหรือไมโครเฟรม หรืออาจจะต้องแบ่งข้อมูลออกเป็นส่วน ๆ แล้วส่งไปกับเฟรมหรือไมโครเฟรมหลายๆ ชุด ขึ้นอยู่กับการจัดเวลาของโฮสทรานแซคชัน และความเร็วในการตอบของอุปกรณ์

เนื่องจากการส่งถ่ายข้อมูลทุกครั้งจะต้องมีการใช้เส้นทางร่วมกัน ดังนั้นทรานแซคชันแต่ละชุดจะต้องมีแอดเดรสของอุปกรณ์ใส่ลงไปด้วย อุปกรณ์ทุกตัวจะมีแอดเดรสของมันเองที่ไม่ซ้ำกับตัวอื่นซึ่งถูกกำหนดโดยโฮส ทรานแซคชันแต่ละชุดเริ่มต้นขึ้นเมื่อโฮสส่งบล็อกข้อมูลซึ่งมีแอดเดรสของอุปกรณ์และตำแหน่งเฉพาะหรือเอนพอยท์ภายในอุปกรณ์ที่ใช้รับข้อมุลรวมอยู่ด้วยทุกสิ่งทุกอย่างที่ส่งออกมาจากอุปการณ์จะเป็นการตอบสนองต่อการร้องขอที่รับเข้ามาจากโฮสเพื่อเป็นการส่งข้อมูลหรือสถานะของอุปกรณ์กลับออกไป



รูปที่ 3.1 แสดงองค์ประกอบของการส่งถ่ายข้อมูล USB

3.1 องค์ประกอบของการส่งถ่ายข้อมูล

การส่งถ่ายข้อมูลแต่ละชนิดของระบบบัสยูเอสบี ประกอบขึ้นจากทรานแซคชั่น(Transaction) ซึ่งแต่ละทรานแซ็คชั่นจะถูกสร้างขึ้นมาจากแพ็กเก็ต โดยที่แต่ละแพ็กเก็ตจะบรรจุข้อมูลเอาไว้ แต่ก่อนจะกล่าวถึงรายละเอียดของทรานแซ็คชั่นนั่นเราจะต้องทำความเข้าใจกับองค์ประกอบพื้นฐานที่สำคัญของการสื่อสารก่อน นั่นคือ เอ็นด์พ้อยน์ และ ไปป์

3.1.1 ดีไวซ์เอนด์พอยน์ (Device Endpoint)

การสื่อสารทุกครั้งข้อมูลจะต้องเดินทางออกมาจากดีไวน์เอ็นด์พ้อยน์หรือถูกส่งเข้าไปในดีไวซ์เอ็นพอยน์ซึ่งเอนด์พอยน์คือบัฟเฟอร์ซึ่งอยู่ภายในตัวคอนโทรลเลอร์ชิป ข้อมูลที่เก็บอยู่ภายในดีไวซ์เอ็นด์พอยน์อาจเป็นข้อมูลที่รับเข้ามา หรืออาจจะเป็นข้อมูลที่รอเพื่อจะส่งออกไป สำหรับทางด้านโฮสนั้นจะไม่มีเอ็นด์พอยน์แต่จะมีบัฟเฟอร์สำหรับเก็บข้อมูลที่รับเข้ามาและสำหรับเก็บข้อมูลซึ่งพร้อมที่จะส่งออกไป โดยโฮสจะทำตัวเป็นเหมือนจุดเริ่มต้นของการสื่อสารกับดีไวซ์เอ็นด์พอยน์

ในข้อกำหนด ยูเอสบี ได้ให้คำจำกัดความของเอ็นด์พอยน์ไว้ว่า "เป็นส่วนหนึ่งของอุปกรณ์ซึ่งสามารถกำหนดตำแหน่งให้ไม่ซ้ำกับอุปกรณ์ตัวอื่นได้ ซึ่งมันสามารถเป็นได้ทั้งแหล่งจ่ายหรือข้อมูลของการสื่อสารที่อยู่ระหว่างโฮสและอุปกรณ์" คำจำกัดความนี้ชี้ให้เห็นได้ว่า การสื่อสารข้อมูลกับเอ็นด์พอยน์นั้นสามารถทำได้ในทิศทางเดียวเท่านั้น แต่สำหรับคอนโทรลเอ็นด์พอยน์ จะเป็นกรณีพิเศษซึ่งมีการสื่อสารในลักษณะสองทิศทาง นั่นหมายความว่าถ้าเรามีคอนโทรลเอ็นด์พอยน์ IN เราก็จะมีคอนโทรลเอ็นด์พอยน์ OUT โดยอัตโนมัติ

เอ็นด์พอยน์แต่ละตัวจะต้องการแอดเดรสที่ไม่ซ้ำกับตัวอื่น ซึ่งแอดเดรสเหล่านี้จะประกอบไปด้วยหมายเลขเอ็นด์พอยน์ และทิศทาง หมายเลขเอ็นด์พอยน์อาจมีค่าอยู่ในช่วง 0 – 15 สำหรับทิศทางจะ

เป็นการมองจากมุมมองของโฮสซึ่ง IN จะมีทิศทางไปยังโฮส และ OUT จะเป็นทิศทางที่ออกจากโฮสเสมอ สำหรับเอ็นด์พอยน์ที่ถูกตั้งค่ามาให้ใช้ในการส่งถ่ายข้อมูลแบบคอนโทรลจะต้องส่งถ่ายข้อมูลทั้งสองทิศทาง ดังนั้นคอนโทรลเอ็นด์พอยน์จึงประกอบขึ้นมาจากคู่ของเอ็นด์พอยน์ IN และ OUT ซึ่งใช้หมายเลขเอ็นด์พอยน์ร่วมกัน นั่นคือ เอ็นด์พอยน์ศูนย์

อุปกรณ์ทุกตัวจะมีเอ็นด์พอยน์ศูนย์ซึ่งถูกตั้งมาให้เป็นคอนโทรลเอ็นด์พอยน์ ตามปกติเรามักใช้คอนโทรลเอ็นด์พอยน์เพียงแค่ตัวเดียวเท่านั้น แต่หากต้องการใช้คอนโทรลเอ็นด์พอยน์มากกว่านั้นก็สามารถทำได้และมีคอนโทรลเลอร์ชิปบางตัวที่สนับสนุนการทำงานในลักษณะนี้

การส่งข้อมูลชนิดอื่นๆ จะสื่อสารข้อมูลในทิศทางเดียวเท่านั้น ซึ่งแอดเดรสของเอ็นด์พอยน์หนึ่งๆ สามารถกำหนดให้เป็นได้ทั้ง IN และ OUT แต่มันจะอยู่แยกกัน เช่น เอ็นด์พอยน์ 1 (EP1) บนตัวอุปกรณ์อาจกำหนดตำแหน่งให้เป็นเอ็นด์พอยน์ IN (EP1 IN) สำหรับส่งถ่ายข้อมูลไปยังโฮส และกำหนดตำแหน่งให้เป็นเอ็นด์พอยน์ OUT (EP1 OUT) สำหรับการส่งถ่ายข้อมูลจากโฮสได้ด้วย

นอกจากเอ็นด์พอยน์ศูนย์แล้ว อุปกรณ์ฟูลสปีดยังสามารถมีเอ็นด์พอยน์เพิ่มเติมได้อีกถึง 30 เอ็นด์พอยน์ (มีค่าแอดเดรส 1 ถึง 15 ซึ่งแต่ละแอดเดรสสนับสนุนทั้ง IN และ OUT) ส่วนอุปกรณ์โลว์สปีดจะมีจำนวนเอ็นด์พอยน์เพิ่มเติมขึ้นมาอีกเพียง 2 เอ็นด์พอยน์เท่านั้นโดยการจับคู่กันด้านทิศทาง เช่น เอ็นด์พอยน์ 1 IN และ เอ็นด์พอยน์ 1 OUT หรือ เอ็นด์พอยน์ 1 IN และเอ็นด์พอยน์ 2 IN

ทุกๆ ทรานแซ็คชั่นที่อยู่บนบัสจะมีการใส่หมายเลขเอ็นด์พอยน์และโค้ดซึ่งเป็นตัวบอกทิศทางการไหลของข้อมูลเอาไว้ หรืออาจมีโค้ดเพื่อบ่งบอกว่าทรานแซ็คชั่นนี้เป็นจุดเริ่มของการส่งถ่ายข้อมูลแบบคอนโทรล โค้ดดังกล่าว ได้แก่ IN , OUT และ SETUP

การให้ชื่อทิศทางของเอ็นด์พอยน์สำหรับทรานแซ็คชั่น IN และ OUT จะยึดเอามุมมองโฮสเป็นหลัก นั่นคือ ในทรานแซ็คชั่น IN ข้อมูลจะเดินทางจากอุปกรณ์รอบข้างไปยังโฮส ส่วนทรานแซ็คชั่น OUT ข้อมูลจะเดินทางจากโฮสไปยังอุปกรณ์รอบข้าง

สำหรับทรานแซ็คชั่น SETUP ข้อมูลจะเดินทางจากโฮสไปยังอุปกรณ์รอบข้างเช่นกัน แต่ทรานแซ็คชั่น SETUP จะเป็นกรณีพิเศษเนื่องจากทรานแซ็คชั่นเป็นตัวเริ่มการส่งถ่ายข้อมูลคอนโทรล อุปกรณ์ ยูเอสบี ทุกตัวต้องรู้จักและเข้าใจทรานแซ็คชั่น SETUP เพื่อให้มันทราบถึงวิธีแปลความหมายของข้อมูลซึ่งบรรจุอยู่ภายใน นอกจากนี้ทรานแซ็คชั่นSETUP ยังเป็นเพียงชนิดเดียวที่อุปกรณ์จะต้องเป็นฝ่ายรับเสมอ ซึ่งการส่งถ่ายข้อมูลชนิดใดๆ อาจใช้ทรานแซ็คชั่น IN หรือ OUT ก็ได้ แต่การส่งถ่ายข้อมูลคอนโทรลจะต้องใช้ทรานแซ็คชั่นSETUP เท่านั้น

ในแต่ละทรานแซ็คชั่นจะมีแอดเดรสของอุปกรณ์และแอดเดรสของเอ็นด์พอยน์บรรจุเอาไว้เมื่ออุปกรณ์ได้รับทรานแซ็คชั่น OUT หรือ SETUP ซึ่งบรรจุแอดเดรสของอุปกรณ์ตัวนั้นเอาไว้ ฮาร์ดแวร์ของมันจะเก็บข้อมูลที่รับเข้าไว้ในเอ็นด์พอยน์ที่เหมาะสม และทำการกระตุ้นอินเทอร์รัพท์ จากนั้นโปรแกรมบริการอินเทอร์รัฟท์ในตัวอุปกรณ์จะประมวลผลข้อมูลที่รับเข้ามาและทำงานต่างๆตาที่ทรานแซ็คชั่นนั้นๆต้องการ ในกรณีที่อุปกรณ์ได้รับทรานแซ็คชั่น IN ซึ่งบรรจุแอดเดรสที่ตรงกับอุปกรณ์ตัว

นั้นและถ้ามันมีข้อมูลที่พร้อมจะส่งไปยังโฮส ฮาร์ดแวร์จะส่งข้อมูลจากเอ็นด์พอยน์ที่ถูกระบุมาลงไปบนบัสและทำการกระตุ้นดินเทอร์รัฟท์ จากนั้นโปรแกรมบริการอินเทอร์รัพท์ที่อยู่ในอุปกรณ์จะทำงานต่างๆ ที่จำเป็นเพื่อให้พร้อมสำหรับทรานแซ็คชั่นIN ถัดไป

3.1.2 ไปป์ (Pipe)

ก่อนที่การถ่ายข้อมูลจะเกิดขึ้น โฮสและอุปกรณ์ต้องสร้างไปป์ขึ้นมาก่อน ไปป์ในระบบบัส ยูเอสบี นั้นไม่ใช่วัตถุที่สามารถจับต้องได้ทางกายภาพแต่มันจะเป็นการเชื่อมต่อทางลอจิกระหว่างโฮสและเอ็นด์พอยน์ ซึ่งใช้ทำงานร่วมกันระหว่างเอ็นด์พอยน์ของอุปกรณ์และซอฟต์แวร์ของโฮสคอนโทรลเลอร์

โฮสจะสร้างไปป์ขึ้นมาในกรณีที่มีการจ่ายไฟให้แก่ระบบ หรือมีการเสียบอุปกรณ์เข้ากับระบบบัสและในกรณีที่มีการร้องขอข้อมูลคอนฟิกิวเรชันจากอุปกรณ์ และโฮสจะทำลายไปป์ที่ไม่ได้ใช้แล้วออกไปเมื่ออุปกรณ์ถูกถอดจากระบบบัส นอกจากนี้โฮสอาจจะร้องขอไปป์ชุดใหม่ หรือทำลายไปป์ที่ไม่ได้ใช้แล้วออกไปที่เวลาใดๆ ได้เมื่อมีการร้องขอคอนฟิกิวเรชันหรืออินเตอร์เฟสชุดใหม่ของอุปกรณ์ อุปกรณ์ยูเอสบีทุกตัวจะมีไปป์ที่เรียกว่าดีฟอลต์คอนโทรลไปป์ (Default Control Pipe) ซึ่งประกอบขึ้นจากเอ็นด์พอยน์ 2 ชุดนั่นคือ เอ็นด์พอยน์ศูนย์ IN และเอ็นด์พอยน์ OUT โดยที่ไปป์นี้จะถูกใช้สำหรับการตั้งค่าการทำงานให้แก่อุปกรณ์

ข้อมูลคอนฟิกิวเรชันที่โฮสรับเข้ามาจะมีดีสคริปเตอร์ของเอ็นพอยน์แต่ละชุดซึ่งอุปกรณ์ต้องการใช้ โดยเอ็นด์พอยน์ดีสคริปเตอร์แต่ละชุดจะใช้แจ้งถึงข้อมูลของเอ็นด์พอยน์นั้นๆ ให้แก่โฮสเพื่อใช้ในการสื่อสารกับมัน ซึ่งข้อมูลนี้จะประกอบด้วยแอดเดรสของเอ็นด์พอยน์ ชนิดของการส่งถ่ายข้อมูล ขนาดสูงสุดของดาต้าแพ็กเก็ต และช่วงเวลาที่ต้องการสำหรับการส่งถ่ายข้อมูล

ในบางกรณีโฮสจะรับคำร้องขอคอนฟิกิวเรชันหลังจากแน่ใจแล้วว่าบัสมีแบนด์วิดธ์ว่างเพียงพอที่จะทำการส่งถ่ายข้อมูลในอัตราที่ต้องการ ซึ่งกรณีนี้จะเกิดขึ้นเมื่อคอนฟิวกิวเรชั่นนั้นร้องขอไปป์สำหรับเป็นสื่อในการส่งถ่ายข้อมูลแบบไอโซโครนัสซึ่งมีการรับประกันอัตราการส่งข้อมูล และการส่งถ่ายข้อมูลแบบอินเทอร์รัพท์ซึ่งมีการรับประกันค่าเวลาสูงสุดระหว่างทรานแซ็คชัน ซึ่งในกรณีนี้โฮสจะทำการตรวจสอบแบนด์วิดธ์ก่อนที่จะสร้างไปป์ขึ้น ถ้าแบนวิดธ์ของระบบบัสมีเพียงพอโฮสจะรับคำร้องขอคอนฟิวกิวเรชั่นนั้น ซึ่งวิธีการนี้ทำให้มั่นใจว่าการส่งถ่ายข้อมูลจะใช้เวลาตามที่มันต้องการแต่ถ้าแบนด์วิดธ์มีไม่เพียงพอโฮสจะปฏิเสธคำร้องขอคอนฟิวกิวเรชันนั้นไป และซอฟแวร์ที่ทำการร้องขอนั้นจะต้องพยายามทำการร้องขอใหม่อีกครั้ง ซึ่งอาจต้องรอจนกระทั่งแบนวิดธ์ของระบบมีพอเพียง หรืออาจเป็นการเลือกคอนฟิกิวเรชันใหม่ซึ่งใช้แบนวิดธ์ที่น้อยกว่า สำหรับไปป์ซึ่งเป็นสื่อสำหรับการร้องขอการส่งถ่ายข้อมูลที่ไม่ต้องการรับประกันทางด้านเวลาที่น้อยกว่า สำหรับไปป์ซึ่งเป็นสื่อสำหรับการร้องขอการส่งถ่ายข้อมูลที่ไม่ต้องการรับประกันทางด้านเวลา โฮสจะไม่ทำการตรวจสอบแบนด์วิดธ์ที่

เหลืออยู่แต่มันเพียงตอบรับว่าจะจัดการส่งถ่ายข้อมูลที่ร้องขอมานั้นลงไปในแบนด์วิดธ์ที่เหลืออยู่ให้ดีที่สุดเท่าที่มันสามารถทำได้

ในข้อกำหนด ยูเอสบี ได้กำหนดชนิดของไปป์ไว้ 2 ชนิด ได้แก่ เมสเสจไปป์ และสตรีมไปป์ซึ่งในการใช้งานจะสอดคล้องกับทิศทางการเดินทางของข้อมูลว่าเป็นหนึ่งหรือสองทิศทาง โดยเมสเสจไปป์เป็นไปป์สองทิศทางซึ่งใช้กับการส่งถ่ายข้อมูลแบบคอนโทรลชนิดเดียวเท่านั้น ส่วนสตรีมไปป์จะเป็นไปป์ที่มีทิศทางเดียวซึ่งใช้กับการส่งถ่ายข้อมูลแบบอื่นๆ นอกเหนือจากการส่งถ่ายข้อมูลแบบคอนโทรล

3.1.2.1 เมสเสจไปป์(Message Pipes)

ในข้อกำหนด ยูเอสบี ได้มีการกำหนดรูปแบบของข้อมูลที่ถูกส่งบนไปป์ชนิดนี้เอาไว้ โดยมันจะถูกควบคุมจากโฮสซึ่งเริ่มต้นด้วยการส่งคำร้องขอจากโฮส จากนั้นข้อมูลจะถูกส่งถ่ายไปในทิศทางที่ต้องการตามที่ได้กำหนดไว้ในคำร้องขอจากโฮส จากนั้นข้อมูลจะถูกส่งถ่ายไปในทิศทางที่ต้องการตามที่ได้กำหนดไว้ในคำร้องขอ เมสเสจไปป์จะอนุญาตให้ข้อมูลไหลได้ทั้งสองทิศทางแต่ไปป์ชนิดนี้จะสนับสนุนการถ่ายโอนข้อมูลแบบคอนโทรลเท่านั้น และดีฟอลต์คอนโทรลไปป์จะเป็นเมสเสจไปป์เสมอ

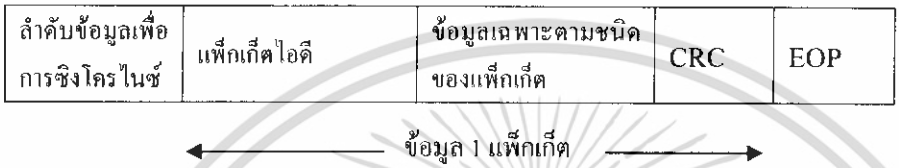
3.1.2.2 สตรีมไปป์(Stream Pipe)

ข้อกำหนด ยูเอสบี ไม่ได้กำหนดรูปแบบของข้อมูลที่สงไปบนไปป์ชนิดนี้เอาไว้ ดังนั้นเราจึงสามารถส่งข้อมูลชนิดใดก็ได้ลงบนสตรีมไปป์ อุปกรณ์ที่เป็นตัวรับจะรับสิ่งที่เดินทางเข้ามา จากนั้นเฟิร์มแวร์ของอุปกรณ์หรือซอฟต์แวร์ของโฮสจะประมวลผลข้อมูลด้วยวิธีการที่เหมาะสมกับงาน

แม้ว่าจะเป็นการสื่อสารด้วยข้อมูลแบบสตรีมก็ตาม แต่อุปกรณ์ซึ่งเป็นตัวรับหรือส่งจะต้องมีการกำหนดรูปแบบเอาไว้ก่อน เช่น แอพพลิเคชั่นซอฟต์แวร์ของโฮสอาจจะกำหนดโค้ดกำหนดโค้ดซึ่งร้องขอให้อุปกรณ์ส่งชุดของไบต์ข้อมูลที่เป็นตัวบอกถึงการอ่านอุณหภูมิและเวลาที่อ่านค่าออกมา แม้ว่าโฮสจะสามารถใช้การส่งถ่ายข้อมูลคอนโทรลกับคำร้องขอที่ผู้ผลิตเป็นผู้กำหนดขึ้นมา

3.2 โครงสร้างการรับส่งข้อมูลระดับล่าง

จากภาพรวมเราทราบกันดีว่าโครงสร้างระดับล่างสุดของยูเอสบี ก็คือแพ็กเก็ต ซึ่งตัวแพ็กเก็ตเองแบ่งออกได้เป็นหลายชนิดกันมาเรียงต่อกันก็จะได้ทรานแซคชันขึ้นมาดังนั้นทรานแซ็กชันก็จะมีหลายชนิดขึ้นอยู่กับชนิดของแพ็กเก็ตที่ประกอบขึ้นมา และเมื่อนำทรานแซ็กชันต่างๆ มาจัดเรียงในเฟรมข้อมูลที่ต้องส่งในทุก 1 มิลลิวินาทีในลำดับต่างๆ กันก็ได้เป็นการส่งข้อมูลชนิดต่างๆ เช่น ถ้าส่งในทุก ๆ เฟรมด้วยขนาดคงที่ก็จะเป็นไอโซโครนัสทรานเฟอร์ แต่ถ้าส่งทุก ๆ คาบเวลาที่แน่นอนแต่ไม่ทุกเฟรมก็เป็นอินเทอร์รัฟต์ทรานเฟอร์



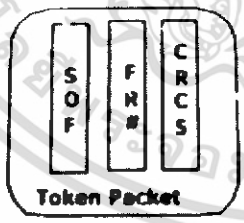
รูปที่ 3.2 รูปแบบของแพ็กเก็ต

ในแต่ละทรานแซคชันประกอบขึ้นจากแพ็กเก็ตต่างๆ ซึ่งโดยส่วนมากจะประกอบด้วย 3 เฟสหรือแพ็กเก็ต แต่บางทรานแซ็คชันบางชนิดอาจจะมีเพียง 2 เฟสหรือมีเพียงเฟสเดียวก็ได้ขึ้นอยู่กับทรานแซ็คชัน

3.2.1 เฟสโทเคน

สามารถแบ่งย่อยออกได้ดังนี้

แพ็กเก็ตเริ่มต้น (SOF:Start Of Frame Token Packet)



รูปที่ 3.3 เฟสโทเคน

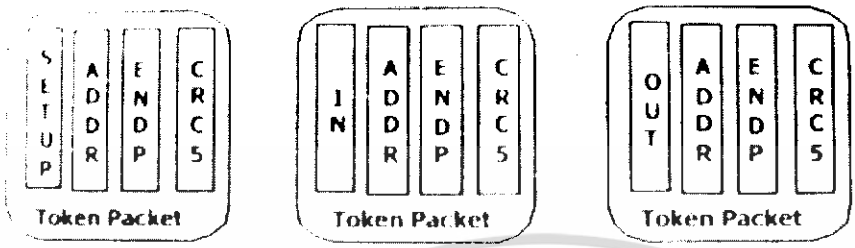
เมื่อเรานำออสซิโลสโคป มาจับในขณะที่ไม่มีการรับส่งข้อมูลบนบัสยูเอสบี พบว่าในทุกๆ 1 มิลลิวินาที โฮสจะส่งแพ็กเก็ตเริ่มต้นออกมาและช่วงระยะเวลาระหว่างแพ็กเก็ตเริ่มต้นแรกกับแพ็กเก็ตเริ่มต้นถัดไปจะเรียกว่าเฟรม (Frame) เมื่อเราใช้คริสตัล 12 เมกกะเฮิร์ท ในการสร้างสัญญาณนาฬิกาจะได้

ระยะเวลา 1 บิตข้อมูล เท่ากับ $\frac{1}{12MHz} = 83.3$ ns

ดังนั้นใน 1 มิลลิวินาที เราสามารถส่งข้อมูลได้ $\frac{1ms}{83.33ns} = 12{,}000$ bit

หรือส่งได้สูงสุด 1,500 ไบต์ซึ่งเป็นค่าทางทฤษฎี ในทางปฏิบัติจะส่งด้วยอัตราการส่งข้อมูลไม่เกิน 1,200 ไบต์ต่อเฟรม

แพ็กเก็ตขาเข้า ขาออก และ Setup แพ็กเก็ต (IN, OUT, SETUP Token Packet)



รูปที่ 3.4 แพ็กเก็ตขาเข้า แพคเก็ตขาออก และ setup แพ็กเก็ต

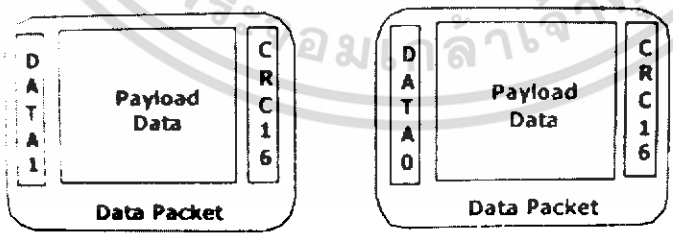
จากรูป ทั้ง3 แพ็กเก็ตจะประกอบด้วยแอดเดรสของอุปกรณ์ขนาด 7 บิต (สามารถต่ออุปกรณ์ได้ 127 ตัว), แอดเดรสของเอ็นพ้อยต์ 4 บิต และ ส่วนตรวจความผิดพลาดของข้อมูลอีก 5 บิต โดยหน้าที่ของแต่ละแพ็กเก็ตเป็นดังนี้

แพ็กเก็ตขาเข้า เป็นการเตรียมที่จะส่งแพ็กเก็ตข้อมูลจากอุปกรณ์ไปยังโฮส

แพ็กเก็ตขาออก เป็นการเตรียมที่จะส่งแพ็กเก็ตข้อมูลจากโฮสไปยังอุปกรณ์ ซึ่งทั้งแพ็กเก็ตขาเข้า และ แพ็กเก็ตขาออก นั้นสามารถส่งไปยังแอดเดรสใดๆ ก็ได้

SETUP แพ็กเก็ต จัดอยู่ในประเภทแพ็กเก็ตขาออก แต่มีลำดับความสำคัญมากกว่า กล่าวคือ ถึงแม้ว่าอุปกรณ์จะปฏิเสธที่จะรับแพ็กเก็ตขาออก แต่ไม่สามารถปฏิเสธ SETUP แพ็กเก็ตได้ ดังนั้น บ่อยครั้งเราจะใช้ SETUPแพ็กเก็ตในการปลุกอุปกรณ์จากการที่อยู่ในโหมดประหยัดพลังงาน

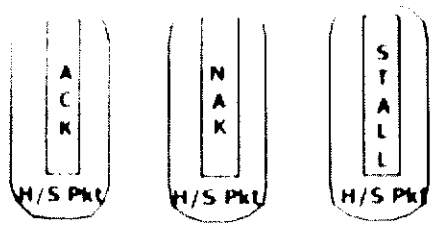
3.2.2 เฟสข้อมูล



รูปที่ 3.5 เฟสข้อมูล

หลังจากที่ส่งแพ็กเก็ต แล้วเราจะส่งแพ็กเก็ตข้อมูล ซึ่งคือข้อมูล (Payload Data) ที่เราต้องการจะส่ง ซึ่งสามารถส่งได้ตั้งแต่ 0-1023 ไบต์โดยจัดรูปแบบเป็น DATA1 กับ DATA0 เรียกว่า ดาต้าท็อกเกิล (Data toggle) ซึ่งจะอธิบายภายหลัง

3.2.3 เฟสตรวจสอบ



รูปที่ 3.6 เฟสตรวจสอบ

เป็นแพ็กเก็ตสำหรับใช้ในการบอกสภาวะของข้อมูลด้านฝั่งรับที่ตอบกลับไปยังฝั่งที่ส่งซึ่งมีอยู่ 3 ชนิด คือ

ACK : Acknowledge Handshake Packet ใช้สำหรับตอบกลับไปว่าได้รับข้อมูลเรียบร้อยแล้ว

NCK : Negative Acknowledge Handshake Packet ใช้สำหรับตอบกลับไปว่ายังไม่สามารถรับข้อมูลตอนนี้ได้เนื่องจากเกิดปัญหา เช่น อาจมีสาเหตุมาจากบัฟเฟอร์ไม่พอ เป็น แพ็กเก็ตปฏิเสธนี้จะเกิดกับทางฝั่งรับเท่านั้นและสามารถปฏิเสธการรับข้อมูลได้ทุกทรานแซ็คชัน ยกเว้น SETUP แพ็กเก็ต แต่โฮสจะไม่มีทางตอบกลับเป็นแพ็กเก็ตปฏิเสธ เพราะว่าโฮสถูกทำให้มีเวลาและทรัพยากรมากพอเสมอ

STALL : Stall Handshake Packet ใช้ในการตอบกลับว่าเกิดปัญหาขึ้นกับอุปกรณ์เช่น อาจจะไม่เข้าใจคำสั่งที่โฮสส่งมา เป็นต้น

3.3 การประกอบแพ็กเก็ตเป็นทรานแซ็คชัน

ทรานแซ็คชั่น คือ การจัดแพ็คเกจของข้อมูลที่จะส่ง โดยสามารถแบ่งทรานแซ็คชั่นออกเป็น 4 ชนิดตามเวลาและคุณภาพ ดังตาราง

ชนิด	สิ่งที่ให้ความสำคัญ	ขนาด(ไบต์)	ตัวอย่าง
Interrupt	คุณภาพ	64 (8 สำหรับ LS)	Mouse ,Keyboard
Bulk	คุณภาพ	64	Printer , Scanner
Isochronous	เวลา	1024	Speaker , Video
Control	คุณภาพ และ เวลา	64 (8 สำหรับ LS)	System Control

ตารางที่ 3.1 เป็นทรานแซ็คชัน ชนิดต่างๆ

เมื่อแบ่งตามความสำคัญของเวลา ข้อมูลที่จะส่งจากโฮสจะถูกจองสัดส่วนไว้ในทุกๆเฟรม สำหรับการส่งข้อมูลแบบ Isochonous และ Control Transfer

และเมื่อแบ่งตามคุณภาพเรามั่นใจได้เลยว่าข้อมูลที่ได้จาก Interrupt , Bulk จะได้ข้อมูลที่ครบถ้วนแน่นอนแต่อาจจะใช้เวลามากกว่า

3.4 ชนิดการส่งถ่ายข้อมูลในระบบบัสยูเอสบี

ข้อกำหนดยูเอสบี ได้กำหนดรูปแบบการส่งถ่ายข้อมูลไว้ 4 ชนิด ได้แก่ การส่งถ่ายข้อมูลแบบคอนโทรล, บัลก์,อินเทอร์รัพท์, และไอโซโครนัส ซึ่งแต่ละชนิดจะมีจุดประสงค์การใช้งานที่แตกต่างกัน

3.4.1 การส่งถ่ายข้อมูลแบบคอนโทรล เป็นการส่งถ่ายข้อมูลเพียงชนิดเดียวซึ่งมีฟังก์ชันที่ถูกกำหนดโดยข้อกำหนดยูเอสบี การส่งถ่ายข้อมูลแบบนี้จะทำให้โฮสสามารถอ่านข้อมูลเกี่ยวกับอุปกรณ์ ตั้งค่าแอดเดรสของอุปกรณ์ เลือกคอนฟิกิวเรชัน และตั้งค่าอื่นๆ ได้ อุปกรณ์ ทุกตัวจะต้องสนับสนุนการส่งถ่ายข้อมูลชนิดนี้

3.4.2 การส่งถ่ายข้อมูลแบบบัลก์ ถูกออกแบบมาเพื่อใช้กับงานที่ไม่คำนึงถึงอัตราการส่งถ่ายข้อมูลเป็นหลัก เช่นการส่งไฟล์ข้อมูลไปยังพรินเตอร์หรือการรับข้อมูลจากสแกนเนอร์ แต่จะเป็นงานที่ต้องการความถูกต้องเป็นสิ่งสำคัญ สำหรับกรณีที่จำเป็นข้อมูลที่ส่งถ่ายด้วยวิธีการนี้จะต้องสามารถหยุดเพื่อรอการส่งถ่ายข้อมูลได้ เช่น ถ้าในระบบบัสมีการขนส่งข้อมูลหนาแน่นมากเนื่องจากมีการส่งถ่ายข้อมูลแบบอื่นๆ ซึ่งมีการรับประกันอัตราการส่งถ่ายข้อมูลอยู่ การส่งถ่ายข้อมูลบัลก์อาจต้องหยุดการส่งถ่ายข้อมูลชั่วขณะเพื่อรอให้บัสว่างลง สำหรับอุปกรณ์ที่สนับสนุนการส่งถ่ายข้อมูลบัลก์นี้จะมีเพียงแค่อุปกรณ์แบบฟูลสปีดและไฮสปีดเท่านั้น อุปกรณ์ USB ทุกตัวอาจไม่ต้องการการสนับสนุนการส่งถ่ายข้อมูลบัลก์ก็ได้

3.4.3 การส่งถ่ายข้อมูลแบบอินเทอร์รัพท์ จะใช้สำหรับอุปกรณ์ซึ่งต้องรับสัญญาณจากโฮสหรือ การเรียกร้องความสนใจของอุปกรณ์เป็นช่วงเวลาที่สม่ำเสมอ นอกเหนือจากการส่งถ่ายข้อมูลแบบคอนโทรลแล้วการส่งถ่ายข้อมูลอินเทอร์รัพท์จะเป็นวิธีหนึ่งซึ่งอุปกรณ์แบบโลว์สปีดสามารถใช้ในการส่งถ่ายข้อมูลได้ คีย์บอร์ดและเมาส์จะใช้การส่งถ่ายข้อมูลอินเทอร์รัพท์ในการส่งสัญญาณการกดคีย์และข้อมูลตำแหน่งของการเคลื่อนเมาส์ การส่งถ่ายข้อมูลแบบอินเทอร์รัพท์สามารถใช้กับโหมดการสื่อสารที่ความเร็วใดๆ ก็ได้ทั้งนี้อุปกรณ์ยูเอสบีทุกตัวไม่จำเป็นต้องสนับสนุนการส่งถ่ายข้อมูลชนิดนี้

3.4.4 การส่งถ่ายข้อมูลแบบไอโซโครนัส เป็นการส่งถ่ายข้อมูลที่มีการประกันเวลาในการส่งแต่จะไม่มีการแก้ไขความผิดพลาดที่เกิดขึ้น ข้อมูลซึ่งเหมาะสำหรับการส่งข้อมูลชนิดนี้ได้แก่ ไฟล์ข้อมูลเสียงซึ่งต้องมีการใช้งานในแบบเวลาจริง การส่งถ่ายข้อมูลชนิดนี้เป็นเพียงชนิดเดียวที่ไม่สนับสนุนการส่งซ้ำอย่างอัตโนมัติในกรณีที่การส่งข้อมูลเกิดความผิดพลาด ดังนั้นความผิดพลาดที่อาจเกิดขึ้นบ้างจึง

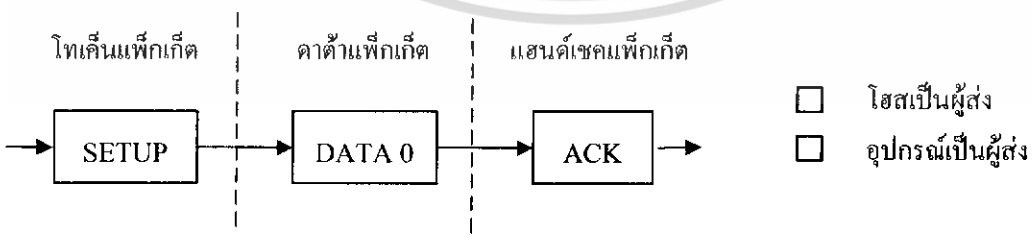
เป็นสิ่งที่ต้องยอมรับได้ อุปกรณ์ที่สนับสนุนการส่งถ่ายข้อมูลไอโซโครนัสจะมีเพียงอุปกรณ์แบบฟูลสปีดและไฮสปีดเท่านั้น ซึ่งอุปกรณ์ยูเอสบีทุกตัวไม่จำเป็นต้องสนับสนุนการส่งถ่ายข้อมูลชนิดนี้

เนื่องจากโครงงานนี้ใช้การส่งถ่ายข้อมูลเพียงสองประเภทคือคอนโทรล และ บัลก์ จึงขออธิบายการส่งถ่ายข้อมูลอย่างละเอียดเพียงแค่สองชนิดเท่านั้น

3.5 การส่งถ่ายข้อมูลชนิดคอนโทรล

การส่งถ่ายข้อมูลชนิดนี้เป็นการสื่อสารแบบสองทิศทาง ซึ่งใช้สำหรับการตั้งค่าการทำงาน การส่งถ่ายคำสั่งหรือข้อมูลสถานะ การส่งถ่ายข้อมูลชนิดนี้จะเริ่มต้นขึ้นจากโฮส และใช้ความพยายามในการส่งมากที่สุด นอกจากนี้ยังมีการตรวจสอบความผิดพลาดด้วยการใช้ CRC(Cyclic Redundancy Check) เพื่อไม่ให้เกิดข้อมูลผิดพลาดได้เนื่องจากข้อมูลที่ใช้การส่งถ่ายข้อมูลชนิดนี้มีความสำคัญที่สุด ขนาดของข้อมูลซึ่งถูกใส่ลงไปในฟิลด์ข้อมูลของดาต้าแพ็กเก็ตหรือเรียกทับศัพท์ว่าดาต้าเพย์โหลด (Data Payload) สำหรับการส่งถ่ายข้อมูลคอนโทรลในอุปกรณ์โลว์สปีดจะมีขนาดเท่ากับ 8 ไบต์, อุปกรณ์ฟูลสปีดมีขนาดเท่ากับ 64 ไบต์ และอุปกรณ์ไฮสปีดสามารถมีขนาดของดาต้าเพย์โหลดเท่ากับ 8,16,32 หรือ 64 ไบต์ ส่วนขั้นตอนการทำงาน หรือสเตจของมันอาจประด้วย 2 ถึง 3 สเตจ ได้แก่ เซ็ตอัพสเตจ, ดาต้าแพคเกต(อาจมีหรือไม่มีก็ได้) และเซ็ตอัพสเตจ

3.5.1 เซ็ตอัพสเตจ เป็นขั้นตอนที่คำร้องขอถูกส่งไปประกอบด้วยแพ็คเกจ 3 ชุด โดยโทเค็นแพ็กเกจจะถูกส่งออกไปเป็นอันดับแรกซึ่งภายใจประกอบด้วย PID ที่มีค่าเป็น เซ็ตอัพ จากนั้นดาต้าเพย์โหลดจะถูกส่งออกมาเป็นลำดับถัดไปซึ่ง พีไอดี ของแพ็คเกจนี้จะเป็น ดาต้า 0 เสมอ ภายในแพ็กเก็ตนี้ประกอบด้วยเซ็ตอัพแพ็กเก็ตซึ่งมีข้อมูลเกี่ยวกับคำร้องขอ รวมถึงหมายเลขของคำร้องขอด้วย ส่วนแพ็กเก็จสุดท้ายคือแฮนด์เช็คแพ็กเก็ตซึ่งใช้สำหรับการบอกสถานะของการรับส่งข้อมูลว่าสำเร็จหรือเปล่า ถ้าอุปกรณ์รับข้อมูลสำเร็จ (CRC และ PID ถูกต้อง) มันจะตอบกลับด้วย ACK เพียงอย่างเดียว มิฉะนั้นมันจะละทิ้งข้อมูลนั้นไปและไม่ส่งแฮนด์เชคกลับไปยังโฮส ทั้งนี้อุปกรณ์จะไม่มีการส่ง STALL หรือ NAK ในการตอบสนองต่อเซ็ตอัพแพ็กเก็ต



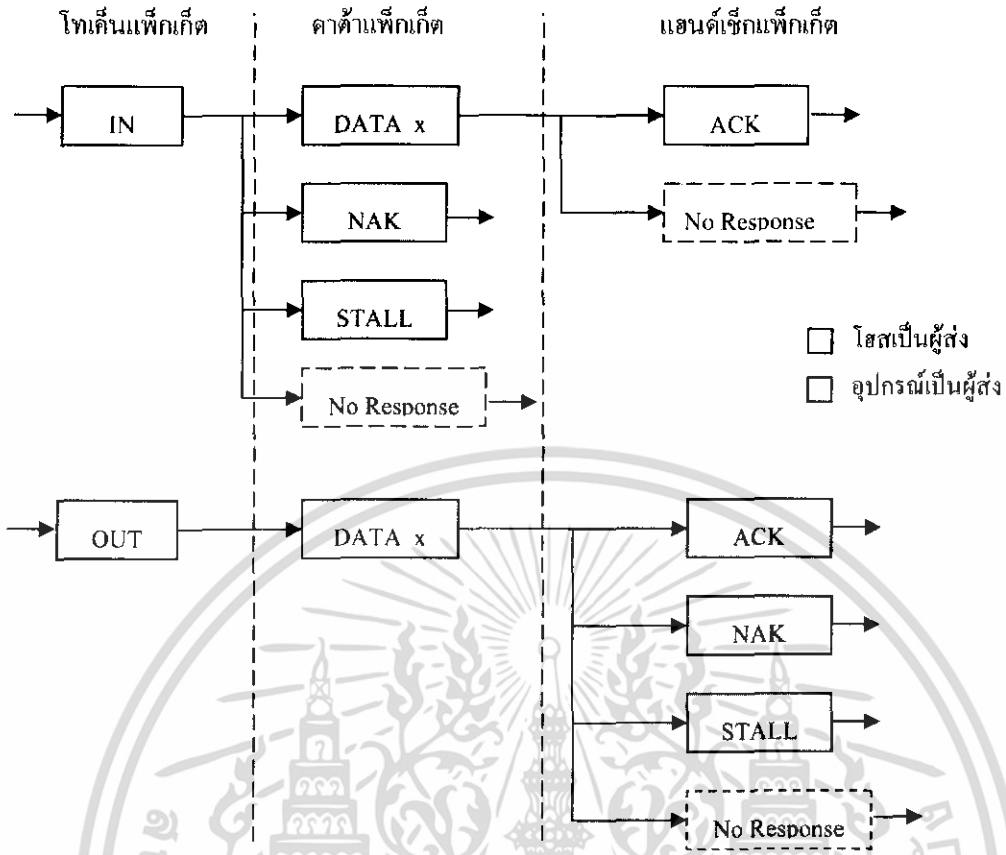
รูปที่ 3.7 เซ็ตอัพสเตจของการส่งถ่ายข้อมูล

3.5.2 ดาต้าสเตจ สเตจนี้อาจมีหรือไม่ก็ได้ โดยมันจะประกอบไปด้วยการส่งถ่ายข้อมูลแบบ อิน หรือ เอ๊าท์ อย่างใดอย่างหนึ่งเป็นจำนวน 1 ครั้งหรือมากกว่านั้น ซึ่งจำนวนของข้อมูลที่ถูกส่งไปในสเตจ

นี้จะถูกกำหนดไว้ในเซ็ตอัพแพ็กเก็ต ถ้าหาข้อมูลมีขนาดเกินกว่าขนาดสูงสุดของดาต้าเพย์โหลด ข้อมูลนั้นจะถูกส่งด้วยการส่งถ่ายข้อมูลหลายครั้ง โดยในแต่ละครั้งจะมีขนาดเท่ากับขนาดสูงสุดของดาต้าเพย์โหลดยกเว้นการส่งครั้งสุดท้าย ดาต้าสเตจจะมีบทบาททีแตกต่างกัน 2 อย่างโดยขึ้นอยู่กับทิศทางของการส่งถ่ายข้อมูล ได้แก่

อิน เกินขึ้นในการส่งถ่ายข้อมูลแบบคอนโทรล Read โดยเมื่อโฮสพร้อมรับข้อมูลคอนโทรล โฮสจะส่งโทเค็น อิน ออกมา ถ้าอุปกรณ์ได้รับโทเค็น อิน ที่มีความผิดพลาด เช่น PID กับส่วนอินเวอร์สของมันไม่สอดคล้องกัน อุปกรณ์จะละทิ้งแพ็กเก็ตนี้ไป แต่ถ้าโทเค็นที่ได้รับมาถูกต้อง อุปกรณ์อาจตอบกลับมาด้วยดาต้าแพ็กเก็ตซึ่งบรรจุข้อมูลคอนโทรลที่จะส่ง หรืออาจเป็นแพ็กเก็ต STALL เพื่อเป็นการบอกว่าเกิดการผิดพลาดขึ้นที่เอนด์พอยน์ หรืออาจเป็นแพ็กเก็ต NAK เพื่อเป็นการบอกโฮสว่าเอนด์พอยสามารถทำงานได้ตามปกติแต่ในขณะนั้นไม่มีข้อมุลที่จะทำการส่ง

เอ๊าท์ เกิดขึ้นในการส่งถ่ายข้อมูลแบบคอนโทรล write โดยเมื่อโฮสต้องการส่งคอนโทรลแพ็กเก็ตไปยังอุปกรณ์ มันจะส่งโทเค็น out ออกมาแล้วตามด้วดาต้าแพ็กเก็ตซึ่งบรรจุซึ่งบรรจุดาต้าเพย์โหลดที่เป็นข้อมูลทคอนโทรลเอาไว้ ถ้าบางส่วนของโทเค็น out หรือดาต้าแพ็กเก็ตไม่ถูกต้อง อุปกรณ์จะละทิ้งไป ถ้าเอนด์พอยด์บัฟเฟอร์ของอุปกรณ์ว่างและสามารถป้อนข้อมุลเข้าไปในเอนพอยท์บัฟเฟอร์ได้ อุปกรณ์จะส่ง ACK ออกมาเพื่อแจ้งให้โฮสทราบว่ามันข้อมูลได้แล้ว ถ้าเอนพอยท์บัพเฟอร์ไม่ว่างเนื่องจากการประมวลแพ็กเก็ตก่อนหน้านี้ยังไม่เสร็จสมบูรณ์ อุปกรณ์จะส่งค่า NAK กลับไป แต่อย่างไรก็ตามมันอาจส่งค่า STALL กลับออกมาถ้าเอนท์พอยท์มีข้อผิดพลาดและบิต HALT เซ็ตอยู่

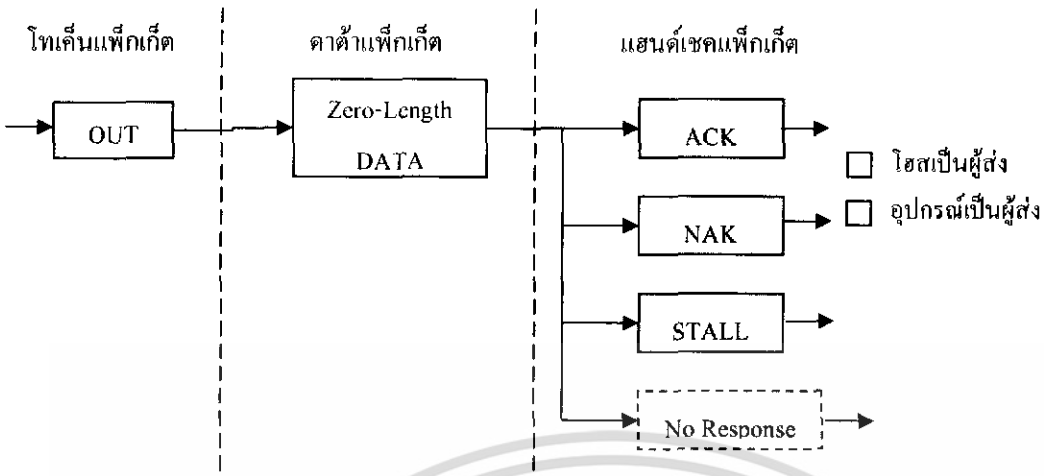


รูปที่ 3.8 ดาต้าสเตจของการส่งถ่ายข้อมูล

3.5.3 สเตตัสสเตจ เป็นสเตจที่ใช้รายงานสถานะของการร้องขอทั้งหมด และมีทิศทางแปรเปลี่ยนไปตามการเปลี่ยนแปลงทิศทางของการส่งถ่ายข้อมูล ซึ่งการรายงานสถานะจะกรทำจากอุปกรณ์เสมอ

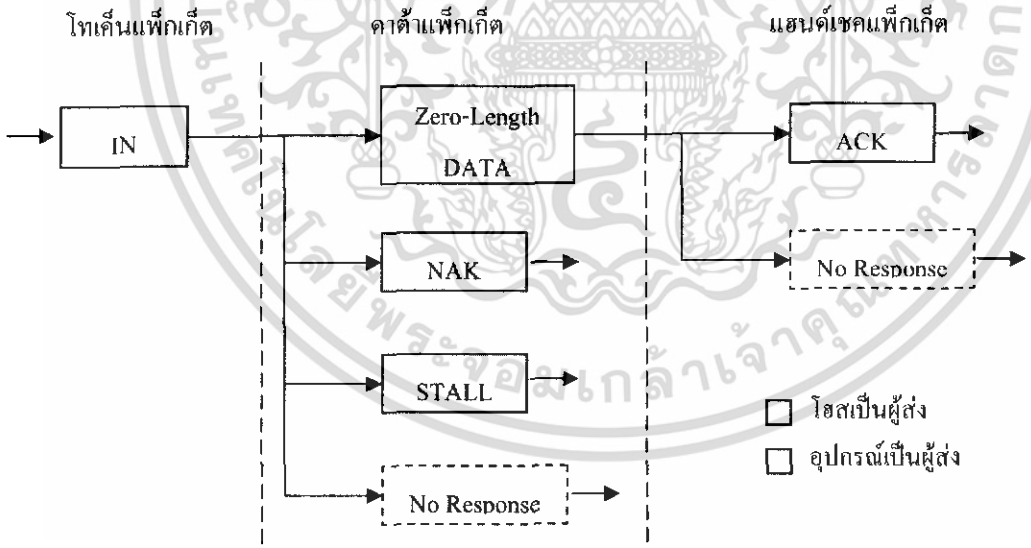
- ถ้าส่งถ่ายครั้งนั้น ๆ ไม่มีดาต้าสเตจ อุปกรณ์จะตอบกลับไปด้วย ACK แต่ถ้าข้อมูลคอนโทรลครั้งนั้นไม่มีดาต้าสเตจ อุปกรณ์จะตอบกลับไปด้วย ACK แต่ถ้าข้อมูลมีความผิดพลาด มันจะละทิ้งข้อมูลนั้นไปและไม่ส่งแฮนด์เช็กแพ็กเก็ตกลับไปยังโฮส

- กรณีเป็นคอนโทรล Read โฮสจะต้องตอบรับ ว่าการรับข้อมูลนี้ทำได้สำเร็จ โดยโฮสต้องทำการส่งโทเค็น out แล้วตามด้วยดาต้าแพ็กเก็ตที่มีความยาวเป็นศูนย์ ในขณะนี้อุปกรณ์จะสามารถงานสถานะของมันลงมาในแฮนด์เช็กสเตจด้วย ACK ซึ่งเป็นการบอกว่าอุปกรณ์ได้ทำตามคำสั่งนี้อุปกรณ์จะส่ง stall ออกมาอย่างไรก็ตามถ้าอุปกรณ์ยังคงประมวลต่อไปมันก็จะส่ง NAK กลับออกมาเพื่อแจ้งโฮสเพื่อทำสเตตัสสเตจอีกครั้งในภายหลัง



รูปที่ 3.9 สเตตัสสเตจของการส่งถ่ายข้อมูลคอนโทรล Read

- กรณีที่เป็นคอนโทรล write อุปกรณ์จะตอบรับว่าการรับข้อมูลทำได้สำเร็จโดยการส่งแพ็กเก็ตที่มีความยาวเป็นศูนย์ในการตอบสนองโทเค็น in อย่างไรก็ตามถ้ามีข้อผิดพลาดเกิดขึ้นมันจะส่ง stall ออกมา หรือ ถ้าอุปกรณ์ยังคงอยู่ในระหว่างการประมวลผลอยู่ มันจะส่ง nak ออกไปเพื่อบอกโฮสให้เข้าสเตตัสสเตจอีกครั้งภายหลัง



รูปที่ 3.10 สเตตัสสเตจของการส่งถ่ายข้อมูลคอนโทรล Write

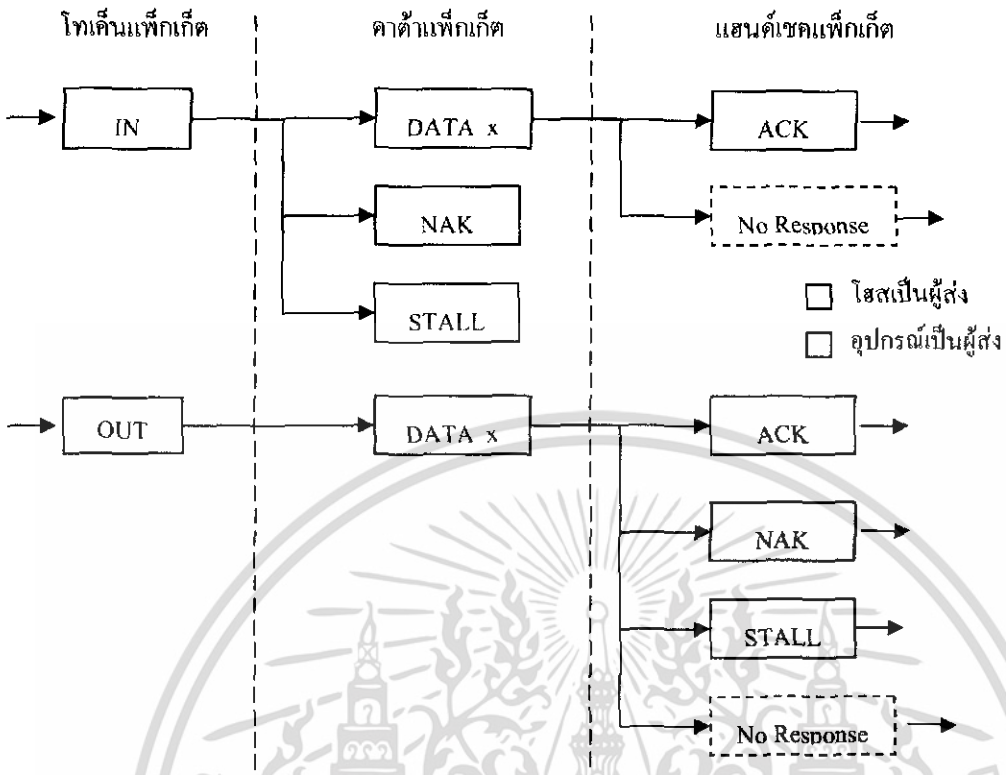
3.6 การส่งถ่ายข้อมูลแบบบัลก์

การส่งถ่ายข้อมูลแบบบัลก์สนับสนุนเฉพาะกับอุปกรณ์แบบฟูลสปีดและไฮสปีดเท่านั้นสำหรับขนาดสูงสุดของแพ็กเก็ตในโหมดฟูลสปีดจะมีขนาดเท่ากับ 8,16,32 หรือ 64 ไบต์ และสำหรับขนาดสูงสุดของแพ็กเก็ตในโหมดไฮสปีดสามารถมีค่าเท่ากับ 512 ไบต์

ทรานแซคชันของบัลก์ อาจประกอบไปด้วยการส่งถ่ายข้อมูล อิน และ เอาท์ ดังนี้

อิน เมื่อโฮสพร้อมรับข้อมูลบัลก์มันจะส่งโทเค็น อิน ออกมา ถ้าอุปกรณ์ได้รับโทเค็น อิน แล้วเกิดความผิดพลาดมันจะละทิ้งแพ็กเก็ตนี้ไป แต่ถ้าอุปกรณ์ได้รับโทเค็นอย่างถูกต้องมันอาจจะทำการตอบกลับด้วยแพ็กเก็ตนี้ไป แต่ถ้าอุปกรณ์นได้รับโทเค็นอย่างถูกต้องมันอาจทำการตอบกลับด้วยแพ็กเก็ต ดาต้า ซึ่งบรรจุข้อมูลบัลก์ที่ต้องการส่ง หรืออาจจะส่งแพ็กเก็ต stall เพื่อเป็นการบอกว่ามีการผิดพลาดขั้นที่เอนพอยน์ หรืออีกกรณีหนึ่งอุปกรณ์อาจส่งแพ็กเก็ต nak เพื่อบอกแก่โฮสว่าเอนด์พอยน์นั้น ๆ สามารถทำงานได้แต่ไม่มีข้อมูลที่จะส่งในขณะนั้น

เอ๊าท์ เมื่อโฮสต้องการส่งข้อมูลบัลก์ มันจะส่งโทเค็น เอ๊าท์ ออกมาแล้วตามด้วยดาต้าแพ็กเก็ตซึ่งบรรจุข้อมูลบัลก์เอาไว้ ถ้าบางส่วนของโทเค็น เอ๊าท์ หรือดาต้าแพ็กเก็ตเกิดการผิดพลาดขึ้นมาอุปกรณ์จะละทิ้งแพ็กเก็ตนี้ไป ถ้าเอนพอยน์บัฟเฟอร์ของอุปกรณ์มีพื้นที่ว่างและจะมีการป้อนข้อมูลเข้าไปในเอนด์พอยน์บัฟเฟอร์ อุปกรณ์ก็จะส่ง ack กลับมาเพื่อแจ้งแก่โฮสว่าได้ทำการรับข้อมูลสำเร็จแล้ว ถ้าเอนด์พอยน์บัฟเฟอร์ไม่มีพื้นที่ว่างเนื่องจามีการประมวลผลแพ็กเก็ตก่อนหน้านี้ อุปกรณ์ก็จะส่ง nak กลับออกมา แต่ถ้าเอนด์พอยน์เกิดความผิดพลาดขึ้นและทิต halt ของมันถูกเซ็ต อุปกรณ์ก็จะส่ง stall กลับออกมา



รูปที่ 3.11 เฟสของการส่งถ่ายข้อมูลบัลก์

3.7 กระบวนการอินิวเมอร์เรท และดีสคริปเตอร์

กระบวนการอินิวเมอร์เรท เป็นขั้นตอนสำคัญที่สำคัญขั้นตอนหนึ่ง ซึ่งโฮสคอมพิวเตอร์สามารถรู้จักกับอุปกรณ์ทุกตัวที่ต่ออยู่ภายในระบบ องค์ประที่สำคัญของกระบวนการอีนิวเมอร์เรท ได้แก่ คำร้องขอ (Request) และดีสคริปเตอร์ต่างๆ เนื้อหาในส่วนนี้จะช่วยให้เข้าใจถึงกลไกต่างๆ ในการตรวจสอบอุปกรณ์ที่ต่อเข้ามาใหม่ในระบบบัสนับตั้งแต่เริ่มเสียบอุปกรณ์เข้ากับพอร์ตยูเอสบี จนกระทั่งถึงการโหลดดีไวซ์ไดร์ฟเวอร์ที่เหมาะสมให้แก่อุปกรณ์ตัวนั้น ซึ่งกระบวนการอินิวเมอร์เรท จะอธิบายในส่วนต่อไป

ยูเอสบีดีสคริปเตอร์

ดีสคริปเตอร์เป็นโครงสร้างข้อมูลที่อธิบายถึงความสามารถ และวิธีการใช้อุปกรณ์ ยูเอสบีโดยที่อุปกรณ์ทุกตัวจะมีลำดับชั้นของดีสคริปเตอร์ซึ่งเป็นตัวที่ใช้อธิบายข้อมูลให้แก่โฮส เช่น อุปกรณ์ตัวนี้คืออะไร,ใครเป็นผู้ผลิต,เวอร์ชันของข้อกำหนดยูเอสบี ที่อุปกรณ์ตัวนี้สนับสนุน,การตั้งค่าต่างๆ สามารถทำได้กี่ทาง, จำนวนและชนิดของเอนพอยน์ที่ใช้เป็นต้น

ดีสคิปเตอร์ที่มีใช้ในข้อกำหนดยูเอสบี 2..0 ได้แก่

ดีไวซ์ดีสคริปเตอร์

ดีไวซ์ควอลิไฟเออร์ดีสคริปเตอร์

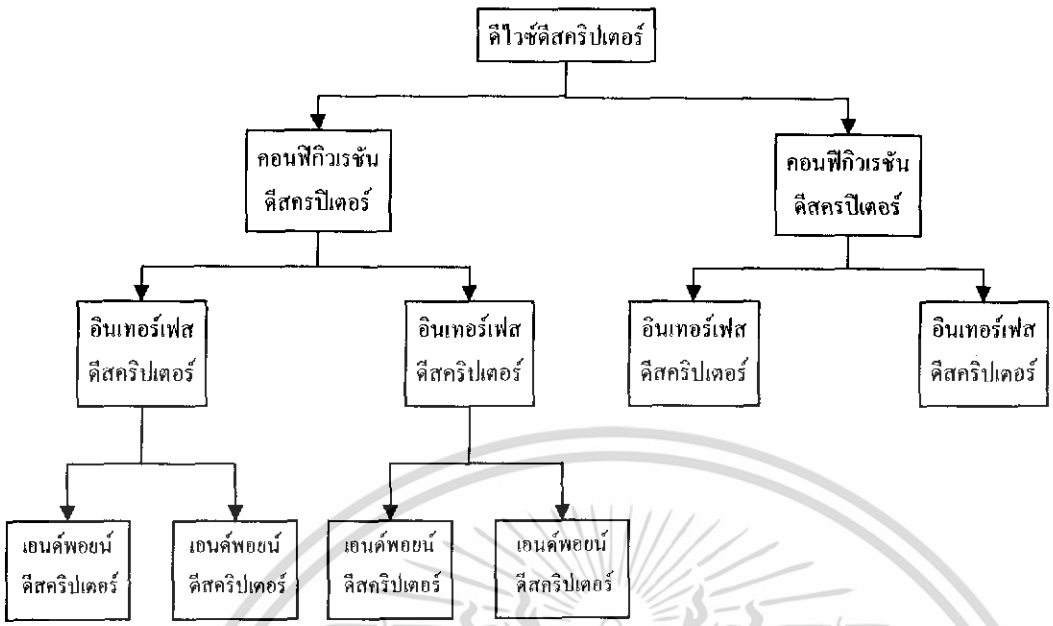
คอนฟิกิวเรชันดีสคริปเตอร์

อัทเธอร์สปีดคอนฟิกิวเรชันดีสคริปเตอร์

อินเตอร์เฟสดีสคริปเตอร์

เอนพอยน์ดีสคริปเตอร์

สตริงดีสคริปเตอร์



รูปที่ 3.12 โครงสร้างของดีสคริปเตอร์

รูปแบบของยูเอสบี ดีสคริปเตอร์สร้างขึ้นจากรูปแบบเดียวกันดังตารางที่ 3.2 โดยไบต์แรกที่เป็นตัวที่ใช้ระบุความยาวของดีสคริปเตอร์ และไบต์ที่สองเป็นตัวที่ใช้บอกชนิดของดีสคริปเตอร์ ถ้าความยาวของดีสคริปเตอร์น้อยกว่าที่กำหนดไว้ในข้อกำหนด ยูเอสบี โฮสจะไม่สนใจดีสคริปเตอร์ตัวนั้นแต่ถ้าความยาวของดีสคริปเตอร์มีค่ามากกว่าที่ต้องการ โฮสจะไม่สนใจในส่วนที่เกินมา และมองหาดีสคริปเตอร์ตัวถัดไปที่อยู่ต่อจากจุดสุดท้ายของความยาวจริงของดีสคริปเตอร์ตัวนั้น

ออฟเซ็ต	ฟิลด์	ขนาด	ค่า	คำอธิบาย
0	bLength	1	ตัวเลข	ขนาดของดีสคริปเตอร์ (ไบต์)
1	bDescriptorType	1	ค่าคงที่	ชนิดของดีสคริปเตอร์
2	bcdUSB	2	BCD	จุดเริ่มต้นของพารามิเตอร์สำหรับดีสคริปเตอร์
...	...	...	...	...

ตารางที่ 3.2 ยูเอสบี ดีสคริปเตอร์

3.7.1 ดีไวซ์ดีสคริปเตอร์

ดีไวซ์ดีสคริปเตอร์เป็นตัวที่ใช้อธิบายพื้นฐานของอุปกรณ์ซึ่งโฮสจะทำการอ่านดีสคริปเตอร์ชุดนี้จากอุปกรณ์ที่ต่อในระบบเป็นชุดแรก ดีไวซ์ดีสคริปเตอร์ประกอบด้วยฟิลด์ทั้งหมด 14 ฟิลด์ ซึ่งเป็นข้อมูลเกี่ยวกับตัวดีสคริปเตอร์เอง, ตัวอุปกรณ์ที่ใช้ดีสคริปเตอร์ชุดนี้อธิบายคอนฟิกิวเรชัน และคลาสของอุปกรณ์

3.7.2 ดีไวซ์ควอลิไฟเออร์ดีสคริปเตอร์

อุปกรณ์ซึ่งอุปกรณ์สนับสนุนทั้งโหมดการทำงานที่ฟูลสปีดและไฮสปีดจะต้องมีดีไวซ์ควอลิไฟเออร์ดีสคริปเตอร์ ถ้าอุปกรณ์สลับความเร็วอาจทำให้ฟิลด์บางฟิลด์ในดีไวซ์ดีสคริปเตอร์เกิดการเปลี่ยนแปลงได้ ดีไวซ์ควอลิไฟเออร์ดีสคริปเตอร์จะใช้สำหรับเก็บฟิลด์ต่างๆ เหล่านี้ซึ่งเป็นของโหมดความเร็วที่ไม่ได้ใช้อยู่ในปัจจุบันเอาไว้ ค่าต่างๆ ที่อยู่ภายในดีไวซ์ดีสคริปเตอร์และดีไวซ์ควอลิไฟเออร์ดีสคริปเตอร์จะมีการสลับกันโดยขึ้นอยู่กับว่าในขณะนั้นความเร็วในถูกเลือกใช้งานอยู่

ดีสคริปเตอร์ชนิดนี้จะมีฟิลด์อยู่ 9 ฟิลด์ ดีสคริปเตอร์จะประกอบไปด้วยข้อมูลเกี่ยวกับตัวดีสคริปเตอร์เอง ตัวอุปกรณ์ที่มันอธิบายอยู่, คอนฟิกิวเรชันและคลาสและคลาสของอุปกรณ์ ฟิลด์ต่างๆ เหล่านี้จะเหมือนกับที่ชั้นดีไวซ์ดีสคริปเตอร์ แต่จะแตกต่างอยู่เพียงแค่มันใช้อธิบายอุปกรณ์ที่ความเร็วซึ่งขณะนั้นไม่ได้ใช้งานเท่านั้น

ค่าเวนเดอร์ไอดีและโพรดักซ์ไอดี, เวอร์ชันของอุปกรณ์ และสตริงเกี่ยวกับผู้ผลิต, ผลิตภัณฑ์ และซีเรียลนัมเบอร์จะไม่มีการเปลี่ยนแปลงดังนั้นในดีไวซ์ควอลิไฟเออร์ดีสคริปเตอร์จึงไม่มีค่าเหล่านี้รวมอยู่ด้วย ต่อไปจะอธิบายแต่ละฟิลด์โดยแบ่งตามกลุ่มหน้าที่ของมัน

คอนฟิกิวเรชันดีสคริปเตอร์

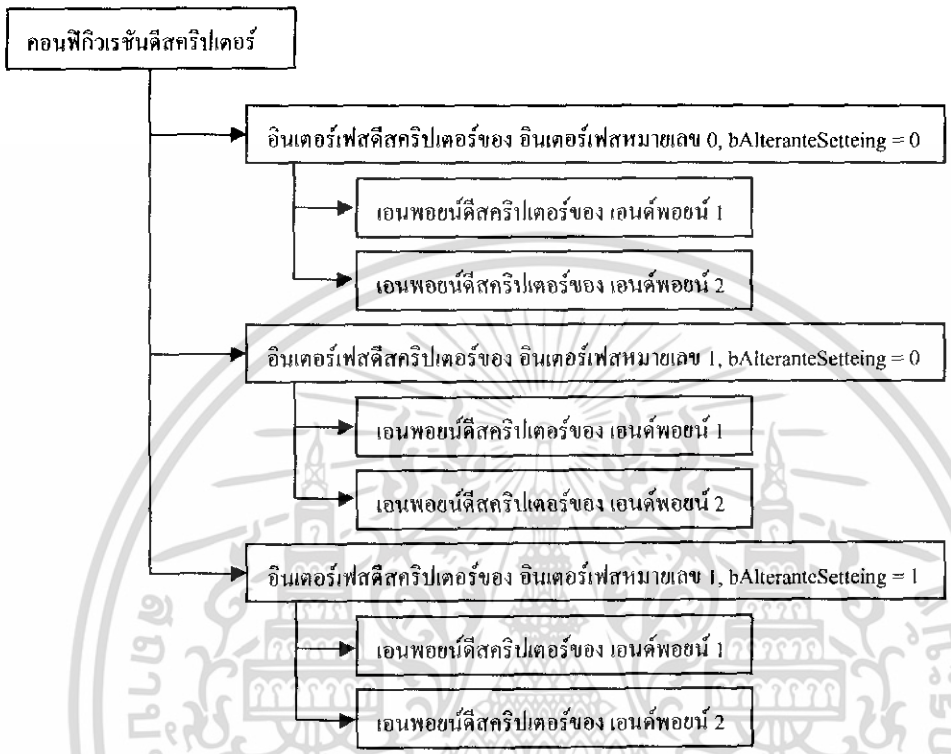
หลังจากที่โฮสได้รับดีไวซ์ดีสคริปเตอร์เข้าไปแล้ว ต่อมาโฮสจะทำการรับคอนฟิกิวเรชันดีสคริปเตอร์, อินเตอร์เฟสดีสคริปเตอร์, และเอนด์พอยน์ดีสคริปเตอร์ของอุปกรณ์

อุปกรณ์แต่ละตัวจะมีคอนฟิกิวเรชันดีสคริปเตอร์อยู่อย่างน้อยหนึ่งชุดซึ่งใช้อธิบายจึงจุดเด่นและความสามารถของอุปกรณ์ ตามปกติคอนฟิกิวเรชันหรือค่าที่ใช้สำหรับการทำงานของอุปกรณ์สามารถมีได้มากกว่า 1 ชุดซึ่งแต่ละชุดจะสนับสนุนโหมดการทำงานที่แตกต่างกัน แต่ในขณะใดขณะหนึ่งจะมีการใช้งานคอนฟิกิวเรชันได้เพียงชุดเดียวเท่านั้น

คอนฟิกิวเรชันแต่ละชุดต้องมีคอนฟิกิวเรชันดีสคริปเตอร์ของมันเอง ซึ่งคอนฟิกิวเรชันดีสคริปเตอร์นี้จะมีข้อมูลเกี่ยวกับการใช้พลังงานของอุปกรณ์และจำนวนอินเตอร์เฟสที่อุปกรณ์ตัวนั้นสนับสนุน นอกจากนี้คอนฟิกิวเรชันดีสคริปเตอร์แต่ละชุดมีดีสคริปเตอร์ย่อยๆ อยู่ภายในตัวมันอีกซึ่งประกอบด้วยอินเตอร์เฟสดีสคริปเตอร์อย่างน้อย 1 ชุด และอาจมีเอนพอยน์ดีสคริปเตอร์เพิ่มอีกด้วย

ทางฝั่งโฮสจะทำการเลือกคอนฟิกิวเรชันโดยใช้คำร้องขอ Set_Configuration และทำการอ่านหมายเลขคอนฟิกิวเรชัยที่ใช้อยู่ในปัจจุบันด้วยคำร้องขอ Get_configuration

เมื่อคอนฟิกิวเรชันดีสคริปเตอร์ถูกอ่านเข้ามา มันจะส่งค่าลำดับชั้นของฟิกิวเรชันทั้งหมดซึ่งจะรวมถึงอินเตอร์เฟสและเอนด์พอยน์ดีสคริปเตอร์ที่เกี่ยวข้องทั้งหมดออกมาดังรูปที่ 3.12 ฟิลด์ wTotalLength จะเป็นตัวที่บอกถึงจำนวนไบต์ทั้งหมดของดีสคริปเตอร์ในลำดับชั้นรวมกัน



รูปที่ 3.13 คอนฟิกิวเรชันดีสคริปเตอร์

คอนฟิกิวเรชันดีสริปเตอร์จะประกอบด้วยฟิลด์ทั้งหมด 8 ฟิลด์ แต่ละฟิลด์จะมีข้อมูลเกี่ยวกับตัวดีสคริปเตอร์, คอนฟิกิวเรชัน และการใช้พลังงานของคอนฟิกิวเรชันนั้นๆ ต่อไปจะอธิบายแต่ละฟิลด์โดยแบ่งตามกลุ่มหน้าที่ของมัน

3.7.3 อัทเธอร์สปีดคอนฟิกิวเรชันดีสคริปเตอร์

ดีสคริปเตอร์ชนิดนี้มีใช้เฉพาะกับอุปกรณ์ซึ่งสนันสนุนทั้งโหมดฟูลสปีดและไฮสปีด และโครงสร้างของอัทเธอร์สปีดคอนฟิกิวดีสคริปเตอร์จะเหมือนกับคอนฟิกิวเรชันดีสคริปเตอร์ ต่างกันเพียงแค่ดีสคริปเตอร์ชนิดนี้ใช้อธิบายคอนฟิกิวเรชันของโหมดความเร็วที่ไม่ได้ใช้อยู่ในปัจจุบัน อัทเธอร์สปีดคอนฟิกิวเรชันดีสคริปเตอร์จะมีดีสคริปเตอร์ย่อยเช่นเดียวกับคอนฟิกิวเรชันดีสคริปเตอร์

3.7.4 อินเตอร์เฟสดีสคริปเตอร์

อินเตอร์เฟสในที่นี้หมายถึงกลุ่มของเอนด์พอยน์ซึ่งถูกใช้โดยฟังก์ชัน ซึ่งอินเตอร์เฟสดีสคริปเตอร์จะมีข้อมูลของเอนพอยน์ซึ่งอินเตอร์เฟสนั้นสนับสนุน

คอนฟิกิวเรชันแต่ละชุดสามารถมีอินเตอร์เฟสได้มากกว่า 1 ชุด ซึ่งแต่ละชุดสามารถใช้งานในเวลาเดียวกันได้อย่างอิสระ สำหรับอินเตอร์เฟสแต่ละชุดนั้นจะต้องมีอินเตอร์เฟสดีสคริปเตอร์ของมันเอง และมีดีสคริปเตอร์ย่อยเป็นเอนพอยน์ดีสคริปเตอร์สำหรับอธิบายเอนด์พอยน์แต่ละตัวซึ่งอินเตอร์เฟสนั้นสนับสนุน

อุปกรณ์ที่มีคอนฟิกิวเรชันซึ่งมีอินเตอร์เฟสหลายชุด โดยที่แต่ละชุดทำงานในเวลาเดียวกันจะเรียกว่า อุปกรณ์คอมโพสิท (Composite Device) ซึ่งในการใช้งานนั้นโฮสจะโหลดดีไวซ์ไดร์ฟเวรอ์สำหรับแต่ละอินเตอร์เฟสขึ้นมางาน

นอกจากนี้ภายในคอนฟิกิวเรชันอาจมีการสนับสนุนอินเตอร์เฟสได้หลายกลุ่ม ซึ่งแยกจากกันเป็นอิสระ โดยแต่ละกลุ่มอาจประกอบไปด้วยอินเตอร์เฟสอีกหลายชุด โฮสจะร้องขอให้มีการเปลี่ยนแปลงไปใช้ในอินเตอร์เฟสอีกกลุ่มหนึ่งโดยการใช้คำร้องขอ Set_Interface และทำการอ่านหมายเลขอินเตอร์เฟสโดยใช้คำร้องขอ Get_interface

3.7.5 เอนพอยน์ดีสคริปเตอร์

เอนพอยน์ดีสคริปเตอร์ใช้ในการอธิบายเอนพอยน์ทุกชุดยกเว้นเอนด์พอยน์ศูนย์ ซึ่งตามปกติอุปกรณ์ทุกตัวจะต้องสนับสนุนเอนพอยน์ศูนย์อยู่แล้ว นอกจากนี้เอนพอยด์ศูนย์ยังถูกสมมติให้เป็นคอนโทรลเอนด์พอยด์และถูกตั้งค่าก่อนที่จะมีการร้องขอดีสคริปเตอร์บางตัวด้วย ส่วนทางด้านโฮสนั้นจะใช้ข้อมูลที่ส่งกลับจากเอนพอยด์ดีสคริปเตอร์เพื่อดูความต้องการทางแบนวิดธ์ของบัสเอนพอยด์ดีสคริปเตอร์ประกอบด้วยขนาดสูงสุดของแพ็กเก็ตและข้อมูลอื่นที่ใช้กับเอนพอยด์

3.7.6 สตริงดีสคริปเตอร์

สตริงดีสคริปเตอร์เป็นส่วนที่ให้ข้อมูลในรูปแบบที่สามารถที่อ่านเข้าใจได้ ดีสคริปเตอร์ตัวนี้เป็นเพียงส่วนเพิ่มเติมเท่านั้นซึ่งถ้าไม่ต้องการใช้งานจะรีเซ็ตค่าฟิลด์ต่างๆ ภายใน ดีไวซ์ดีสคริปเตอร์, คอนฟิกิวเรชันดีสคริปเตอร์ และอินเตอร์เฟสดีสคริปเตอร์ที่อ้างอิงสตริงเตอร์ให้เป็น 0 เพื่อเป็นการบอกว่าไม่มีการใช้สตริงดีสคริปเตอร์

สตริงจะถูกเข้ารหัสในรูปแบบของยูนิโค้ด ซึ่งสามารถสร้างผลิตภัณฑ์ให้รองรับกับการใช้งานในกลายภาษาได้ สตริงแต่ละชุดจะมีค่าอินเด็กซ์ของมันเอง โดยสตริงอินเด็กซ์ 0 จะมีหน้าที่พิเศษโดยตัวมันเอง โดยเป็นตัวส่งรายการหมายเลขของภาษาที่สนับสนุน

บทที่ 4
การออกแบบและทดสอบ

4.1 วัตถุประสงค์

ปัจจุบันอุปกรณ์ต่างๆที่เชื่อมต่อผ่านพอร์ต USB ได้ถูกสร้างและพัฒนาขึ้นจำนวนมาก เช่น Flashdrive, Printer , Mp3player เป็นต้น ซึ่งโดยปกติแล้วอุปกรณ์เหล่านี้จะต้องอาศัยเครื่องคอมพิวเตอร์เป็นตัวกลางในการติดต่อสื่อสารแลกเปลี่ยนข้อมูลระหว่างโฮสคอนโทรลเลอร์ในเครื่องPC ไปยังอุปกรณ์ต่อพ่วงUSB Device ซึ่งอาจเป็นการใช้ทรัพยากรอย่างไม่คุ้มค่าในงานบางลักษณะงาน ที่ไม่ได้ใช้ฟังก์ชันการทำงานอื่นๆของเครื่องPC

โครงงานนี้จึงมุ่งศึกษาถึงการออกแบบสร้างอุปกรณ์โฮสคอนโทรลเลอร์ที่ทำมาทดแทนโฮสคอนโทรลเลอร์ในเครื่องPC ซึ่ง จะเป็นตัวกลางในการเชื่อมต่อโดยตรงระหว่าง อุปกรณ์ต่อพ่วง2ตัว ให้ทำการแลกเปลี่ยนระหว่างกันได้โดยตรง โดยมีโหมดการทำงานสำคัญๆเบื้องต้น อิงตามมาตรฐานของ USB

4.2 ขอบเขตของการศึกษา

เนื่องจากเทคโนโลยีUSB มีขอบเขตที่กว้าง การออกแบบตัวอุปกรณ์โฮสคอนโทรลเลอร์ในโครงงานนี้จึงมุ่งศึกษาและสร้างส่วนของระบบการทำงานและการจัดการด้านโปรโตคอล และการตรวจสอบแก้ไขข้อผิดพลาดระหว่างทำการรับส่งข้อมูล ส่วนในด้านของการแปลงระดับของสัญญาณไฟฟ้าจากตัวสาย การจัดการด้านพลังงาน จะยังไม่ขอศึกษาลงไปในรายละเอียด เพราะในปัจจุบันมีตัวทรานซีฟเวอร์ (USB Transceiver) ที่ผลิตขึ้นมาตามมาตรฐานออกขายจำนวนมาก ซึ่งสามารถนำมาประยุกต์ใช้ในการพัฒนาโครงงานนี้ต่อไปได้

4.3 คุณสมบัติของโฮสคอนโทรลเลอร์ที่ทำการออกแบบ

-มี 2 พอร์ท เพื่อใช้เสียบต่อเข้ากับ อุปกรณ์ต่อพ่วง 2 ตัว ในการแลกเปลี่ยนข้อมูลระหว่างกัน

-มีฟังก์ชันการทำงานและการจัดการโปรโตคอล อิงตามมาตรฐานที่USBกำหนด

-สนับสนุนการส่งถ่ายข้อมูลในโหมด Full speed

-สนับสนุนการส่งถ่ายข้อมูลแบบ Control และ แบบ Bulk โดยมีส่วนตรวจสอบความผิดพลาดของข้อมูล (CRC checking)

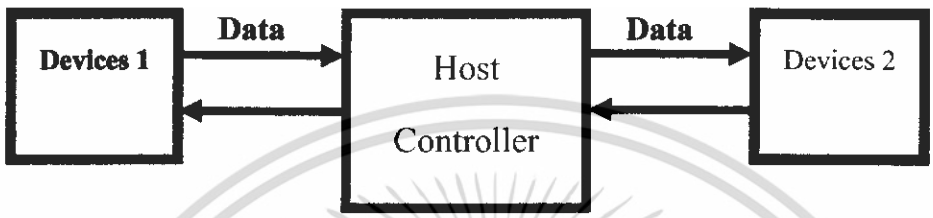
4.4 คุณสมบัติของอุปกรณ์ต่อพ่วงเสมือนที่สร้างขึ้น(USB Device)

-มี 1 พอร์ทสำหรับเชื่อมต่อเข้ากับโฮส เพื่อทำการแลกเปลี่ยนข้อมูลระหว่างกัน

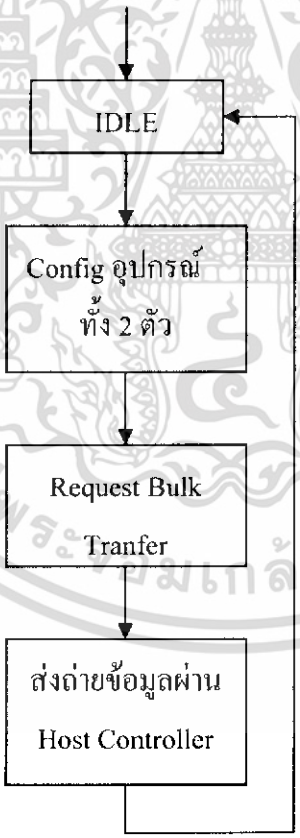
-มีฟังก์ชันการทำงานและการจัดการโปรโตคอล อิงตามมาตรฐานที่USBกำหนด

-สนับสนุนการส่งถ่ายข้อมูลในโหมด Full speed

-มีฟังก์ชันที่สนับสนุนการทำงานที่ถูกโฮสร้องขอมา เพื่อทำการแลกเปลี่ยนข้อมูลระหว่างกัน



รูปที่ 4.1 แสดงลักษณะการเชื่อมต่อกันระหว่างอุปกรณ์ต่อพ่วงกับโฮสคอนโทรลเลอร์ที่ออกแบบ



รูปที่ 4.2 โครงสร้างการทำงานหลักของโฮสคอนโทรลเลอร์ที่ออกแบบ

4.5 ขั้นตอนการอินนิวเมอร์เรทระหว่างอุปกรณ์กับโฮสคอนโทรลเลอร์ที่ออกแบบ

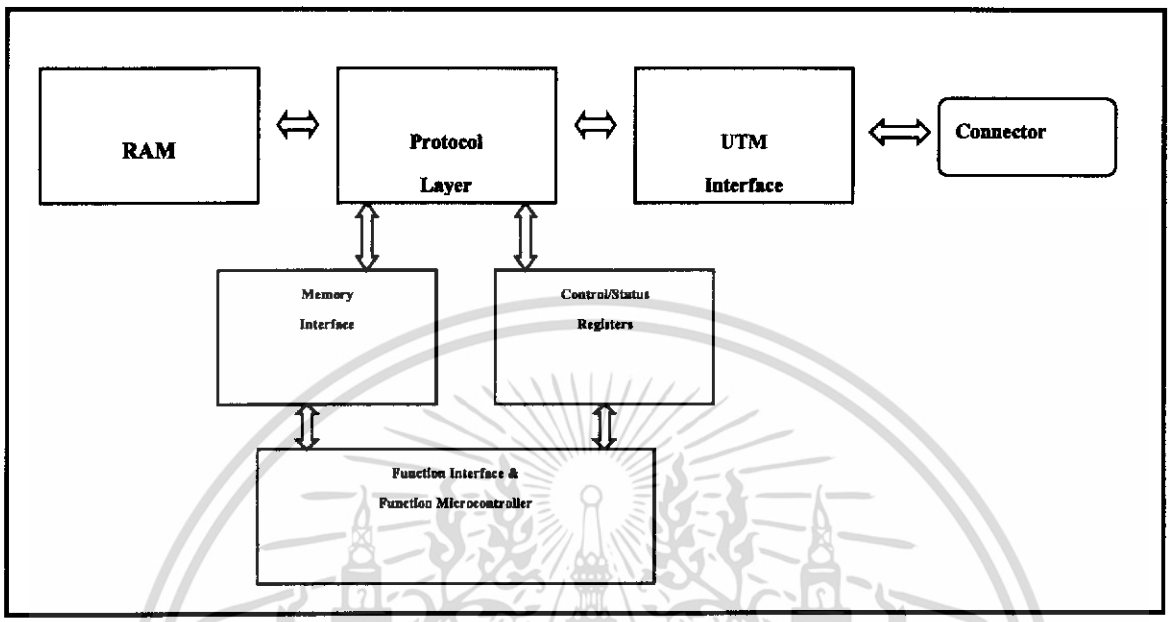
1. เมื่อเสียบอุปกรณ์ USB ดีไวซ์เข้ากับพอร์ต แต่ละพอร์ตของโฮส จากนั้นโฮสทำการจ่ายไฟเลี้ยงให้กับตัวอุปกรณ์ทั้ง2
2. ฟังก์ชันภายในโฮสจะทำการตรวจจับอุปกรณ์ที่มาเชื่อมต่อ มีการตรวจจับแรงดันไฟฟ้าในแต่ละพอร์ต เพื่อเช็คว่าเป็นอุปกรณ์ ไฮสปีท ฟูลสปีด หรือ โลว์สปีด โดยยังมีการจ่ายกำลังงานให้แก่อุปกรณ์ แต่ยังไม่มีการรับส่งข้อมูลใดๆ
3. โฮสทำการรับทราบข้อมูลต่างๆที่เกี่ยวข้องกับตัวอุปกรณ์ที่ต่อพ่วงอยู่ โดยมีฟังก์ชันที่จะคอยรายงานเหตุการณ์ที่เกิดขึ้นบนแต่ละพอร์ต จากนั้นโฮสจะส่งคำร้องขอ Get_Port_Status ออกไปเพื่อหาข้อมูลเพิ่มเติม
4. โฮสทำการรีเซ็ตตัวอุปกรณ์หลังจากที่ทราบข้อมูลเกี่ยวกับอุปกรณ์ตัวใหม่แล้ว โดยโฮสคอนโทรลเลอร์จะส่งคำร้องขอ Set_Port_Feature ไปเพื่อรีเซ็ตอุปกรณ์แต่ละตัว
5. จากนั้นฟังก์ชันในโฮสคอนโทรลเลอร์จะทำการสร้างทางเดินสัญญาณระหว่างอุปกรณ์และบัส โดยหลังจากทำการรีเซ็ตแล้ว ก็จะส่งคำร้องขอ Set_Port_Feature ไปอีกครั้ง เพื่อตรวจสอบภาวการณ์รีเซ็ตว่าเสร็จสมบูรณ์หรือไม่
6. เมื่อผ่านสภาวะรีเซ็ตแล้ว อุปกรณ์จะเข้าสู่สภาวะเริ่มต้นซึ่งพร้อมในการตอบสนองต่อการส่งถ่ายข้อมูลคอนโทรลบนดีฟอลต์ไปป์ที่เอนด์พอยน์ศูนย์ ในขณะนี้อุปกรณ์จะสื่อสารกับโฮสผ่านแอดเดรสเริ่มต้นคือ 00h
7. โฮสส่งคำสั่งร้องขอ Get_Descriptor เพื่อหาขนาดสูงสุดของแพ็กเก็ตบนดีฟอลต์ไปป์ โดยทำการส่งคำร้องขอไปยังแอดเดรส 0 และ เอ็นด์พอยน์ศูนย์
8. โฮสทำการให้แอดเดรสแก่อุปกรณ์ โดยเป็นแอดเดรสจริงที่ใช้ในการติดต่อระหว่างกันที่ไม่ซ้ำกัน โดยโฮสจะส่งคำร้องขอ Set_Address ออกมา อุปกรณ์จะตอบรับคำร้องขอและเก็บค่าแอดเดรสใหม่เข้าไป จนกว่าจะถูกถอดออกจากพอร์ต
9. โฮสอ่านข้อมูลเกี่ยวกับความสามารถของอุปกรณ์ โดยส่งคำร้องขอ Get_Descriptor ไปยังแอดเดรสใหม่เพื่ออ่านดีไวซ์ดีสคริปเตอร์ และ คอนฟิกิวเรชันดีสคริปเตอร์ที่มีทั้งหมด
10. โฮสเลือกการคอนฟิกกิวเรชัน โดยการส่ง Set_Comfiguration ออกไปยังอุปกรณ์ จากนั้นอุปกรณ์จะถูกเซ็ตคอนฟิกิวเรชันและชนิดของการอินเทอร์เฟสจนเสร็จสมบูรณ์ แล้วเริ่มทำการส่งถ่ายข้อมูลตามรูปแบบที่โฮสกำหนด (เริ่มส่งถ่ายข้อมูลแบบบัลก์)

4.6 แนวคิดที่นำไปสู่การออกแบบMini Host Controller

รูปที่ 4.3 แสดงถึงการคิดออกแบบอุปกรณ์ที่จะนำมาทดแทนโฮสคอมพิวเตอร์

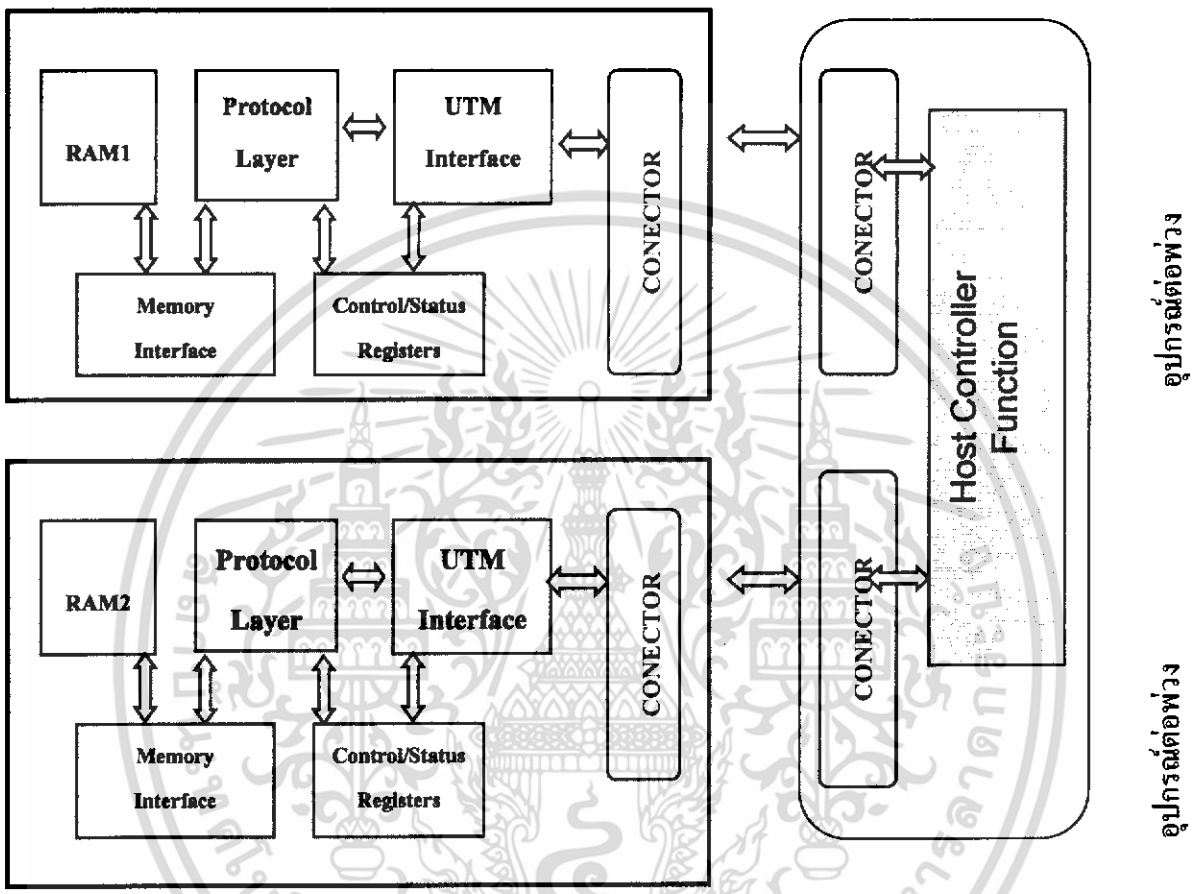
จากวัตถุประสงค์ที่ต้องการสร้างตัวกลางที่ใช้ในการรับส่งข้อมูลระหว่างอุปกรณ์แทนคอมพิวเตอร์ ซึ่งในการใช้การเฉพาะทางนั้น อาจเกินความจำเป็น และ สิ้นเปลือง ดังนั้นจึงนำแนวคิดที่จะทำให้อุปกรณ์ต่อพ่วงมาต่อกับอุปกรณ์ที่จะทำงานออกแบบให้ทำหน้าที่เป็นตัวกลางแทน ซึ่งจะช่วยลดความยุ่งยากและฟังก์ชันที่เกินความจำเป็นออกไปได้ โดยในโครงงานนี้ได้ทำการออกแบบระบบให้สนับสนุนการส่งถ่ายข้อมูลได้ 2 แบบ คือ แบบControl Transfer และ Bulk Transfer

โดยในการออกแบบระบบภายในทั้งหมดของยูเอสบีนั้น ไม่สามารถกระทำได้ทั้งหมดในเวลาอันสั้น จึงจำเป็นต้องเลือกออกแบบระบบในส่วนที่สำคัญและมีประโยชน์ ซึ่งในที่นี้เลือกออกแบบในส่วนของการจัดการด้านโปรโตคอล หรือ โปรโตคอลเลเยอร์ ซึ่งจะทำงานร่วมกับส่วนอื่นๆที่เหลือดังรูป ซึ่งส่วนที่เหลือเหล่านี้ สามารถซื้อเป็นไอซีสำเร็จรูปมาต่อเพิ่มจนเป็นระบบยูเอสบีที่สมบูรณ์ได้ โดยมีรูปดังอย่างระบบภายในของยูเอสบี แสดงได้ดังรูป



รูปที่ 4.4 แสดงโครงสร้างภายในของระบบยูเอสบี

โดยความสำคัญของส่วนระบบจัดการโปรโตคอลที่ออกแบบนี้ จะถูกใช้สำหรับการจัดการข้อมูลต่างๆที่จะทำการรับส่งระหว่างตัวอุปกรณ์ให้เป็นไปตามมาตรฐานยูเอสบี ซึ่งเป็นส่วนที่จำเป็นต้องมีอยู่ทั้งในส่วนของตัวอุปกรณ์ที่มาต่อพ่วงเอง และ ตัวอุปกรณ์โฮสคอลโทรลเลอร์ โดยแสดงได้ดังรูป



รูปที่ 4.5 ส่วนโปรโตคอลเลเยอร์ ที่อยู่ทั้งในตัวอุปกรณ์และโฮสคอนโทรลเลอร์

4.7 การออกแบบส่วนต่างๆภายในโปรโตคอลเลเยอร์

ทำการออกแบบระบบย่อยๆภายในProtocol Layer จะมีส่วนสำคัญต่างๆที่ออกแบบดังนี้คือ

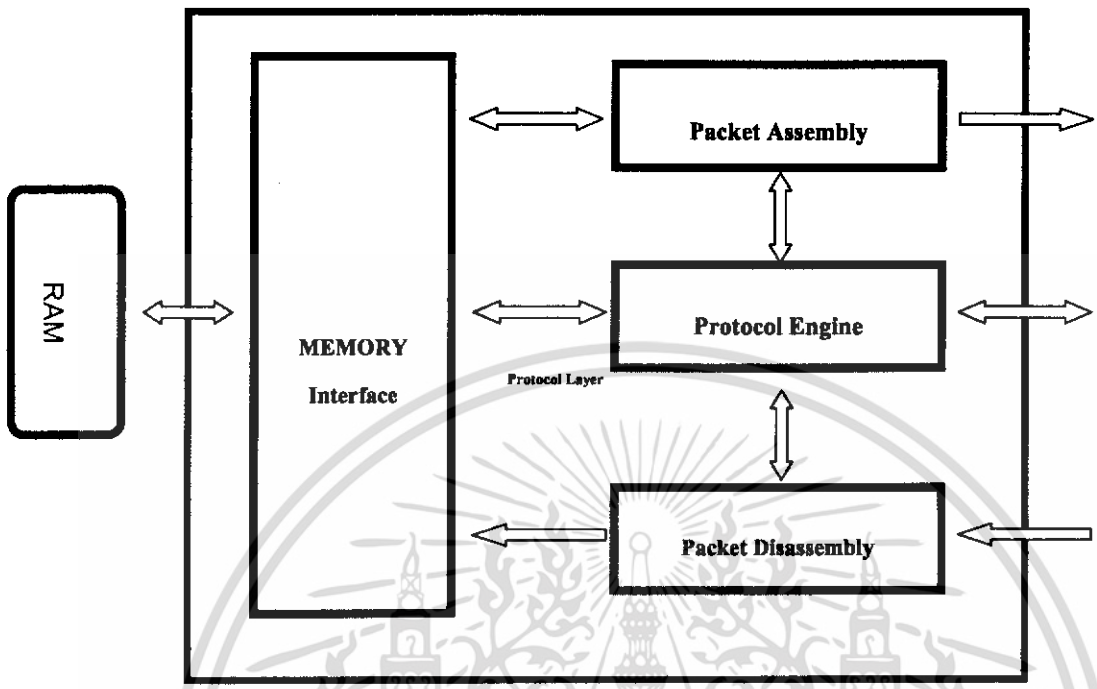
1.ส่วนสร้างแพคเก็ตข้อมูล (Packet Assembly)

2.ส่วนแยกแพคเก็ตข้อมูล (Packet Disassembly)

3.ส่วนระบบจัดการโปรโตคอล (Protocol Engine)

4. ส่วนจัดการติดต่อกับหน่วยความจำ (Memory Interface)

Protocol Layer



รูปที่ 4.6 แสดงโครงสร้างภายในของระบบยูเอสบีในส่วนโปรโตคอลเลเยอร์

1.ส่วนสร้างแพคเก็ตข้อมูล (Packet Assembly)

เป็นส่วนที่ทำการสร้าง Packet ตามฟอร์แมทของ USB แล้วทำการส่งต่อออกไปยังส่วนอื่นๆ ซึ่งมีการใส่ PID และ Checksum หรือ ทำการส่ง ข้อมูล ลงไปเมื่อได้รับการร้องขอจากโฮส

2.ส่วนแยกแพคเก็ตข้อมูล (Packet Disassembly)

เป็นส่วนที่ออกแบบมาเพื่อทำหน้าที่ในการถอดรหัสแยกส่วนPacket ทั้งหมดที่เข้ามา แล้วทำการส่งต่อไปยังส่วนอื่นๆที่เกี่ยวข้อง โดยมีการแยกส่วน PID และ Sequence Number ออกมาแล้วทำการตรวจสอบความถูกต้องของPacketข้อมูลที่รับเข้ามาด้วย ซึ่งทำงานร่วมกับ Protocol Layer

3.ส่วนระบบจัดการโปรโตคอล (Protocol Engine)

เป็นส่วนที่ใช้ในการจัดการกับโปรโตคอลให้เป็นไปตามมาตรฐานUSB ทั้งในส่วนของ Handshake และ ข้อมูล Control ได้แก่ SOF Token, ACK, NACK, IN, OUT, SETUP เป็นต้น

4. ส่วนจัดการติดต่อกับหน่วยความจำ (Memory Interface)

เป็นส่วนที่ทำหน้าที่ในการอินเตอร์เฟส กับMemory ภายนอก ซึ่งจะทำงานร่วมกับส่วนProtocol Engine , Packet Disassembly และ Packet Assembly ซึ่งอยู่ในภายในโปรโตคอลเลเยอร์ เช่นกัน

4.8 การออกแบบและผลการทดลองของระบบภายในของภาคเรียนที่ 1

จากส่วนต่างๆดังกล่าวที่ทำการออกแบบ เริ่มนำระบบย่อยมาทำการเขียนcode ภาษา HDL โดยในส่วนแรกที่หยิบมาทำการทดลองคือ ส่วนตรวจสอบความถูกต้องของข้อมูลในแพคเก็ต (CRC Check) ซึ่งถูกใช้ประกอบเข้าในขั้นตอนการสร้างแพ็คเก็ต และ ถูกตรวจสอบและแยกข้อมูลจริงออก ในขั้นตอนการแยกแพคเก็ต

การคำนวณ CRC

กำหนด G(X) : Polynomial ของ Data

P(X) : Generator polynomial

จะต้องหา Polynomial F(X) ที่เป็น code message (data+BCC)

ที่หารด้วย P(X) ลงตัว

วิธีการ

1. เอา G(X) คูณด้วย XN-K
2. หาร XN-K[G(X)] ด้วย P(X) (ไม่มีการทดและการยืมโดยใช้ Exclusive OR)
3. เอา Remainder C(X) บวก ข้อ 1 ผลลัพธ์คือ

F(X) = XN-K[G(X)] + C(X)

ตัวอย่าง สมมุติข้อมูลที่จะส่ง Data คือ 110011 กำหนด

generator P(X) = 11001

พบว่า G(X) 6 bits, P(X) 5 bits ดังนั้น C(X) 4 bits, N=10, K=6

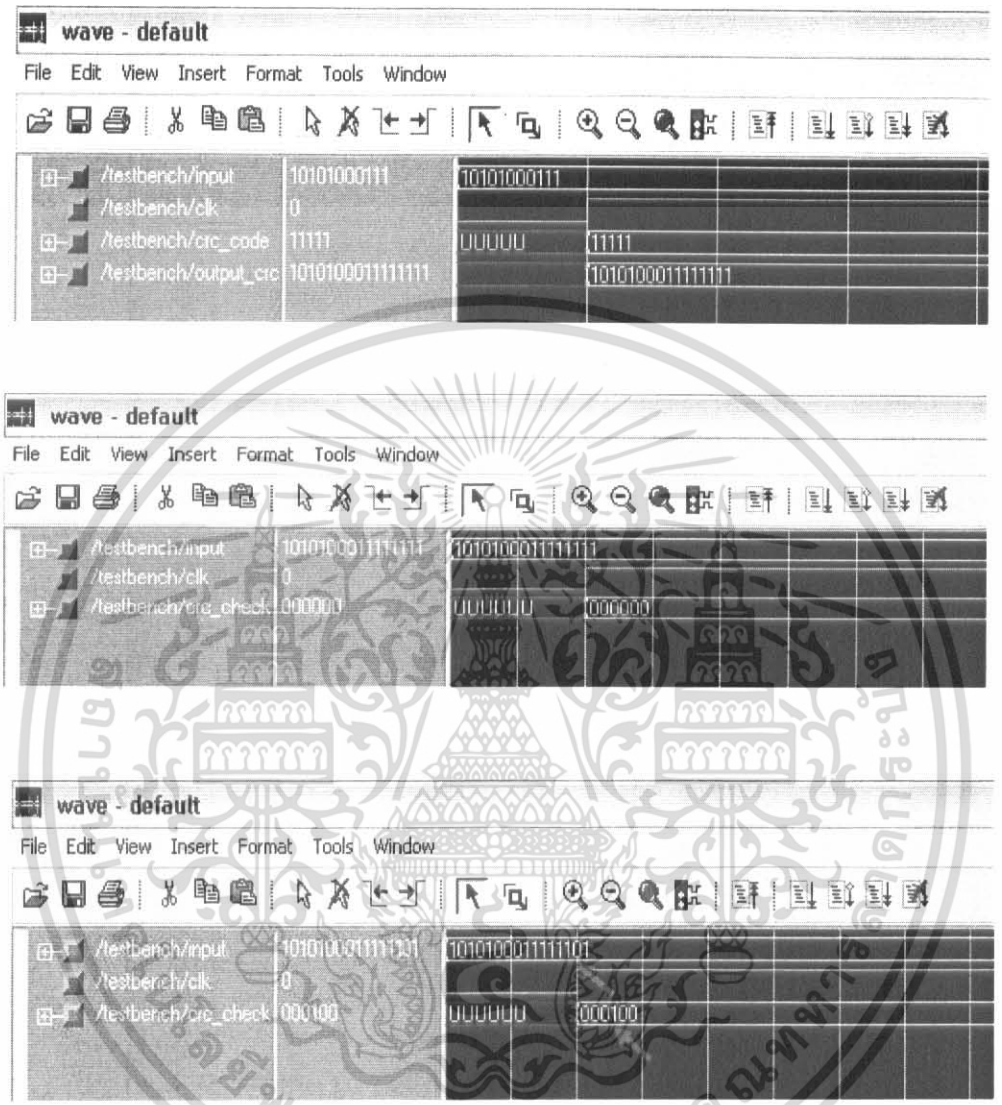
G(X) = 110011 (X5 + X4 + X + 1) P(X) = 11001 (X4 + X3 + 1)

XN-K[G(X)] = X4 .(X5 + X4 + X + 1) = X9 + X8 +X5 + X4 คือ 1100110000

XN-K[G(X)] หารด้วย P(X) คือ 1100110000 หารด้วย 11001

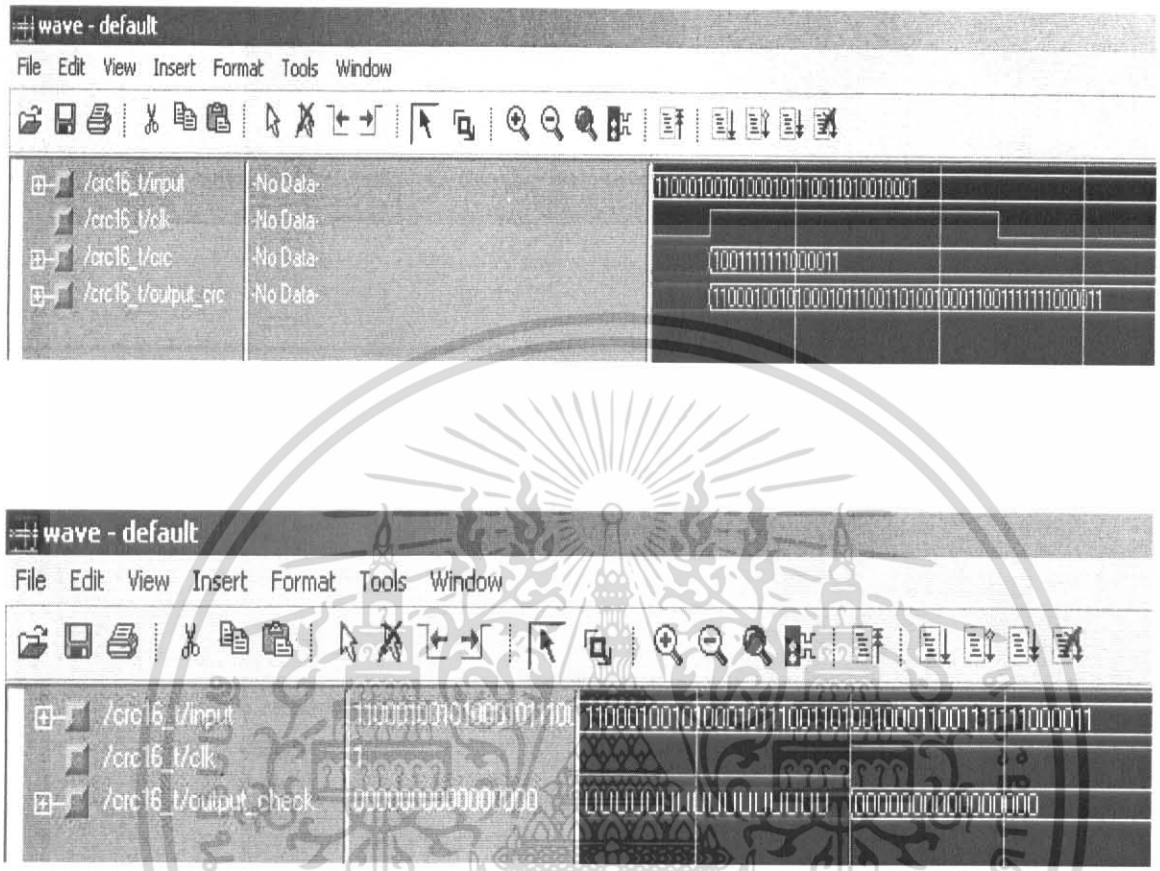
ได้ เศษ(Remainder) = C(X) = 1001

Waveform ผลการทดลองการทำCRC5



รูปที่ 4.7 ผลการทดลองการทำ CRC 5

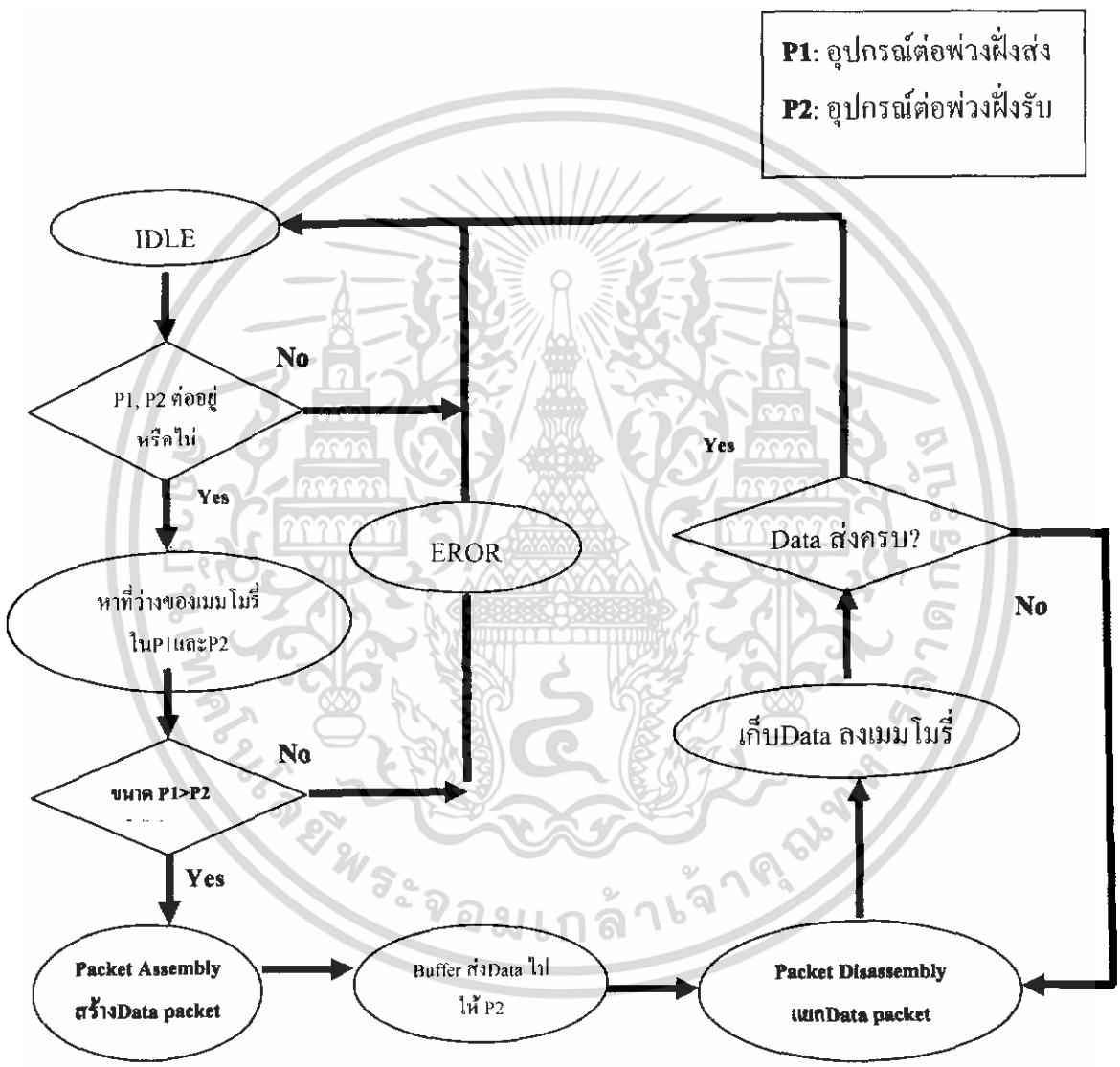
Waveform ผลการทดลองการทำ CRC16



รูปที่ 4.8 ผลการทดลองการทำ CRC 16

4.9 ออกแบบระบบภายในในส่วนที่มีขอบเขตงานในอยู่ในภาคเรียนที่ 2

ทำการออกแบบโครงสร้างการทำงานของตัวระบบใหญ่ โดยให้การกำหนดฝั่งตายตัวของอุปกรณ์ต่อพ่วงทั้งสองฝั่งว่า ให้ฝั่งใดทำหน้าที่เป็นฝั่งส่ง ฝั่งใดทำหน้าที่เป็นฝั่งรับ ทั้งนี้เพื่อให้ง่ายต่อการใช้งานจริง มีกระบวนการดังรูป



รูปที่ 4.9 แผนภาพการทำงานของระบบหลักภายในมินิโฮสคอนโทรลเลอร์

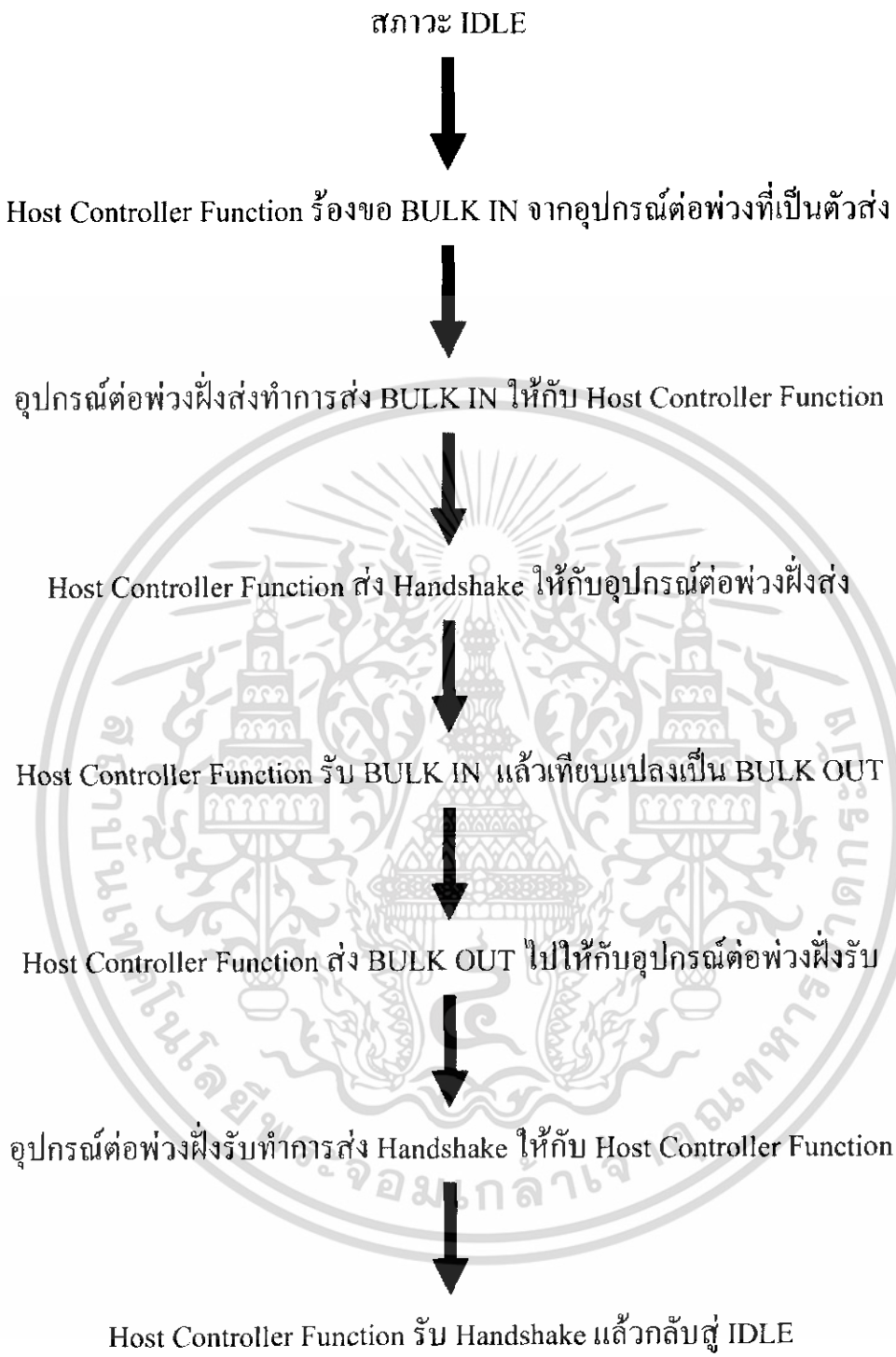
4.9.1 การออกแบบการส่งข้อมูลชนิด คอนโทรล

ใช้ในการเริ่มคอนฟิคระหว่างตัวอุปกรณ์ต่อพ่วงและตัวโฮสคอนโทรลเลอร์ โดยในการออกแบบนี้ ได้ระบบให้อุปกรณ์ที่มาเสียบต่อพ่วงก่อน จะถูกทำการคอนฟิคก่อน แล้วตัวที่เหลือจะถูกคอนฟิคในลำดับถัดไป โดยจะมีขั้นตอนการทำงานสรุปได้ดังนี้

1. Host Controller Function สั่งให้ Protocol Layer ในตัวเองส่ง TOKEN SETUP ไปให้กับอุปกรณ์ตัวแรกที่มาเชื่อมต่อ
2. Protocol Layer ในตัวHost ส่งData0ไปให้ตัวอุปกรณ์เพื่อทำการเริ่มคอนฟิค
3. อุปกรณ์ที่ถูกทำการคอนฟิคตัวแรกรับData0 แล้วทำการส่ง Handshake กลับไปที่โฮส
4. เริ่มเข้าสู่ช่วง Data stage โดยอุปกรณ์ต่อพ่วงจะส่ง Data packet ไปให้กับโฮส
5. โฮสทำการ Handshake กลับมา แล้วทำการระบุค่าแอดแดรสให้แก่ตัวอุปกรณ์ตัวแรกที่เข้ามาเชื่อมต่อ และเก็บข้อมูลเฉพาะต่างๆของตัวอุปกรณ์กลับไป
6. Host Controller Function สั่งให้ Protocol Layer ในตัวเอง ส่งTOKEN SETUP ไปให้อุปกรณ์ต่อพ่วงตัวที่เหลือ
7. Protocol Layer ในโฮสส่ง Data0 ไปให้กับอุปกรณ์ตัวที่เหลือ
8. อุปกรณ์ต่อพ่วงตัวที่เหลือทำการส่ง Handshake กลับมายังโฮสว่าได้รับข้อมูลแล้ว
9. เข้า Data stage โดยอุปกรณ์ต่อพ่วงจะส่ง Data packet ไปให้กับโฮส
10. โฮส ทำการส่งHandshake กลับ และทำการระบุแอดเดรสให้กับอุปกรณ์ต่อพ่วงตัวที่เหลือนั้น และเก็บข้อมูลเฉพาะของตัวอุปกรณ์ส่งกลับไป

4.9.2 การออกแบบการส่งข้อมูลชนิด บัลก์

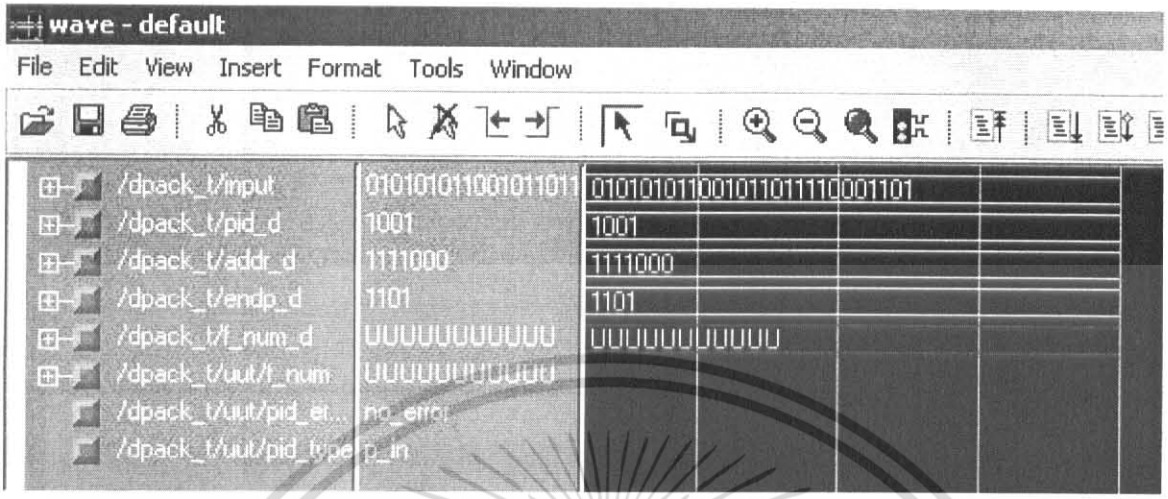
การส่งข้อมูลแบบบัลก์ในที่นี้จะทำการออกแบบให้ตัวอุปกรณ์ต่อพ่วงและโฮสคอนโทรลเลอร์สามารถรองรับการส่งข้อมูลชนิดนี้ภายหลังจากการส่งข้อมูลแบบคอนโทรลเพื่อที่จะใช้ทำการส่งผ่านข้อมูลจริงๆที่ถูกตัดแบ่งเป็นดาต้าแพ็คเก็ต ซึ่งได้ทำการออกแบบลำดับขั้นตอนการส่งถ่ายข้อมูลไว้ดังนี้



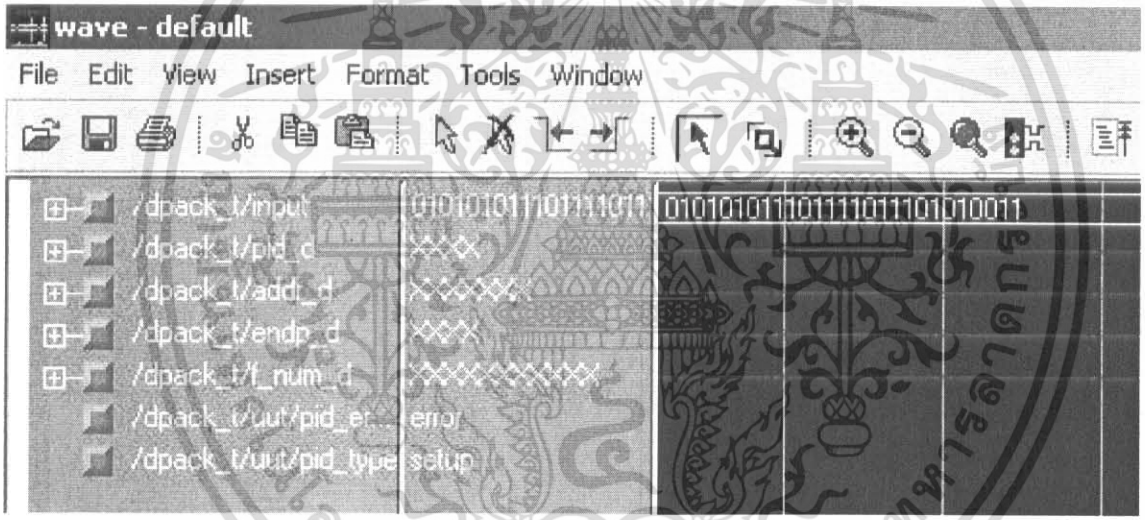
รูปที่ 4.10 ลำดับขั้นตอนการทำงานของมินิโฮสคอนโทรลเลอร์ในโหมดบัลก์

Waveform ผลการทดลองการแยกแพ็คเก็ต(Packet Disassembly)

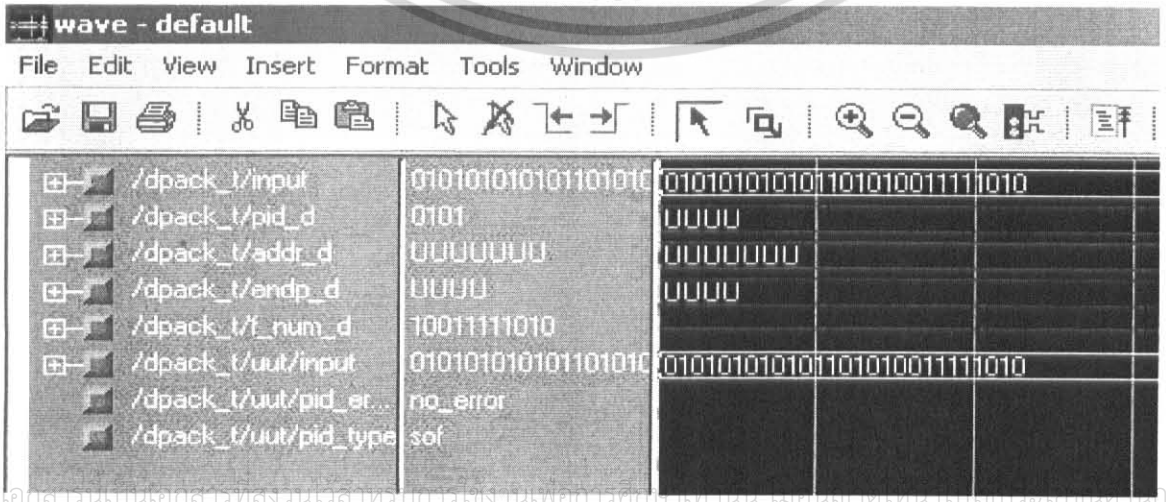
รูปที่ 4.11 การแยกแพ็คเก็ตชนิด Token IN Packet



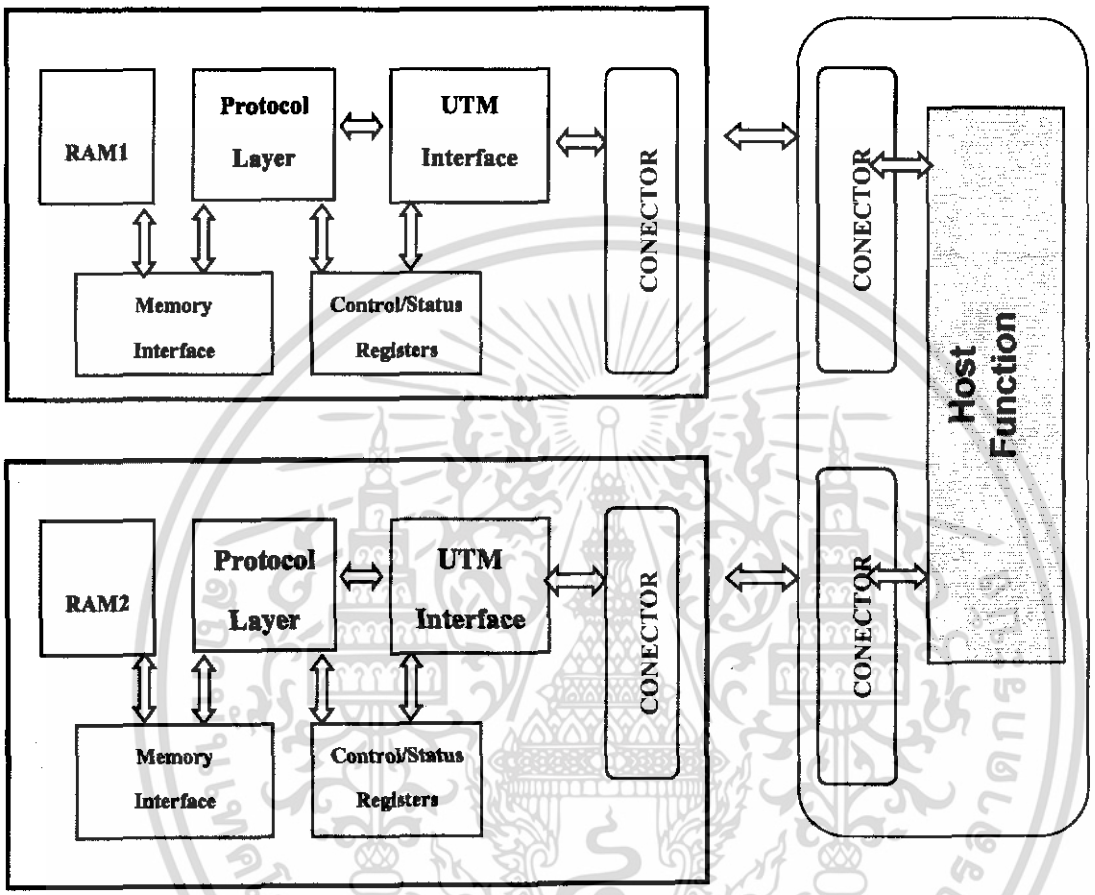
รูปที่ 4.12 การแยกแพ็คเก็ตชนิด Start Of Frame Packet



รูปที่ 4.13 การแยกแพ็คเก็ตชนิด Token Setup Packet



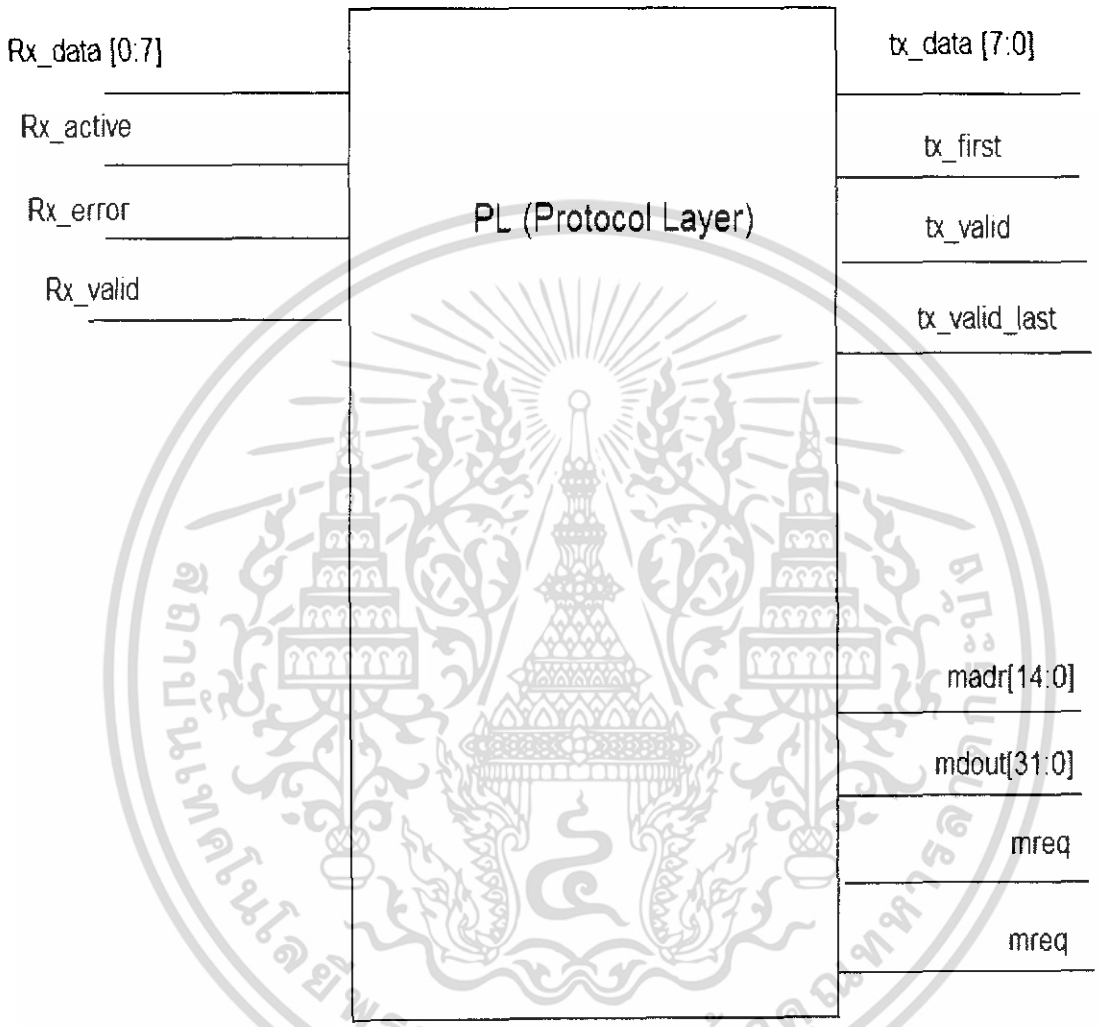
4.9.3 การออกแบบส่วนต่างๆของระบบ



รูปที่ 4.14 แสดงระบบการทำงานทั้งหมดที่เกี่ยวข้อง

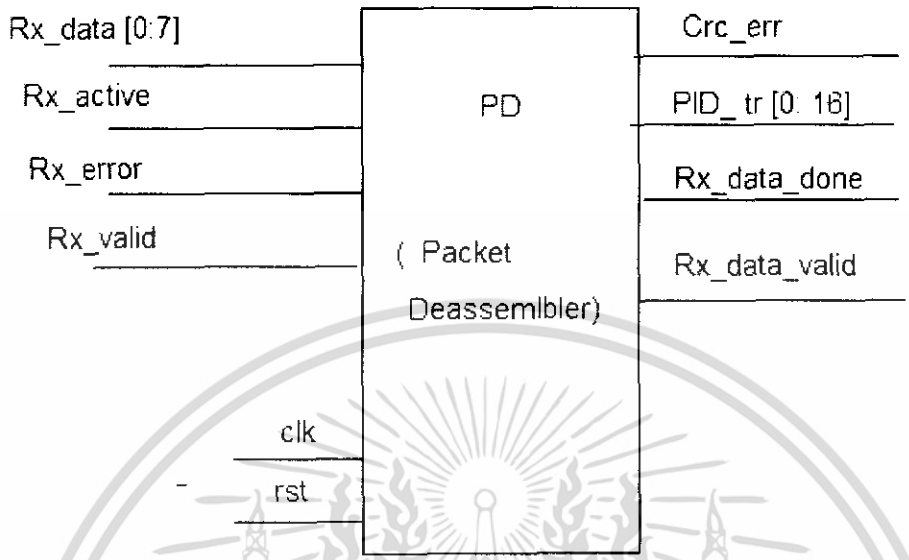
เนื่องจากไม่สามารถหาทรานซีฟเวอร์มาใช้จริงได้ ระบบต่างๆที่สร้างขึ้น จริงไม่สามารถนำมาทดสอบค่ารวมทั้งกระบวนการได้ และเนื่องจากปัญหาเรื่องระยะเวลาที่น้อยกว่าขอบเขตงานที่มีความซับซ้อนมาก จึงไม่สามารถออกแบบสร้างระบบจัดการเม็มโมรี่ได้ได้ทัน ซึ่งในส่วนที่ได้ทำการออกแบบเสร็จสิ้น มีดังนี้

4.9.4 การออกแบบในส่วนโปรโตคอลเลเยอร์ที่ใช้จริง

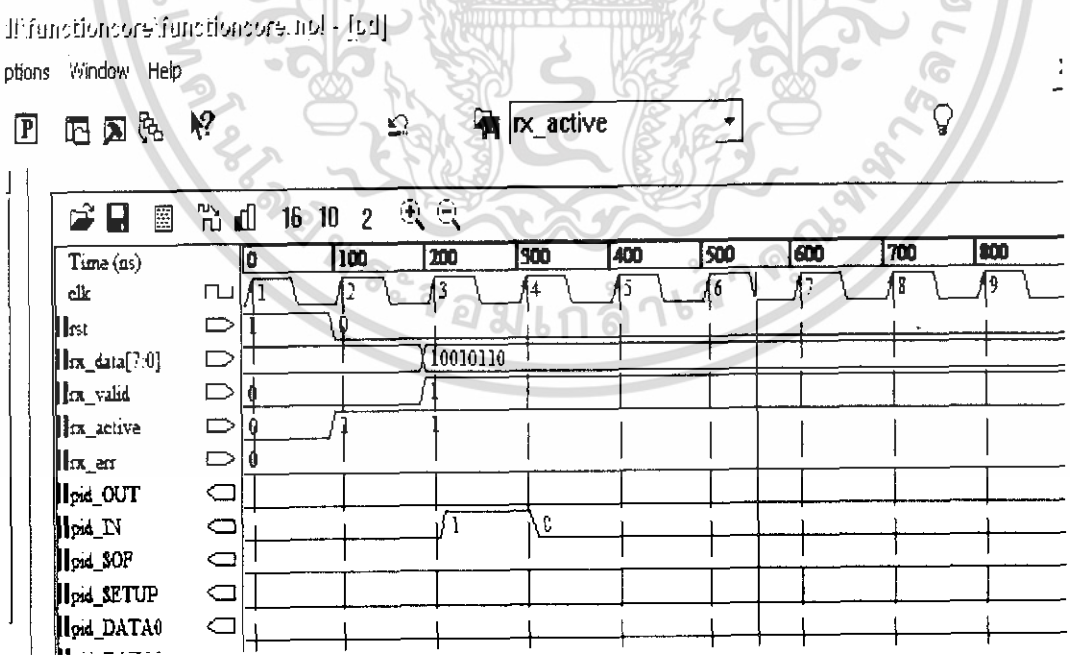


รูปที่ 4.15 การออกแบบในส่วนโปรโตคอลเลเยอร์ที่ใช้จริง

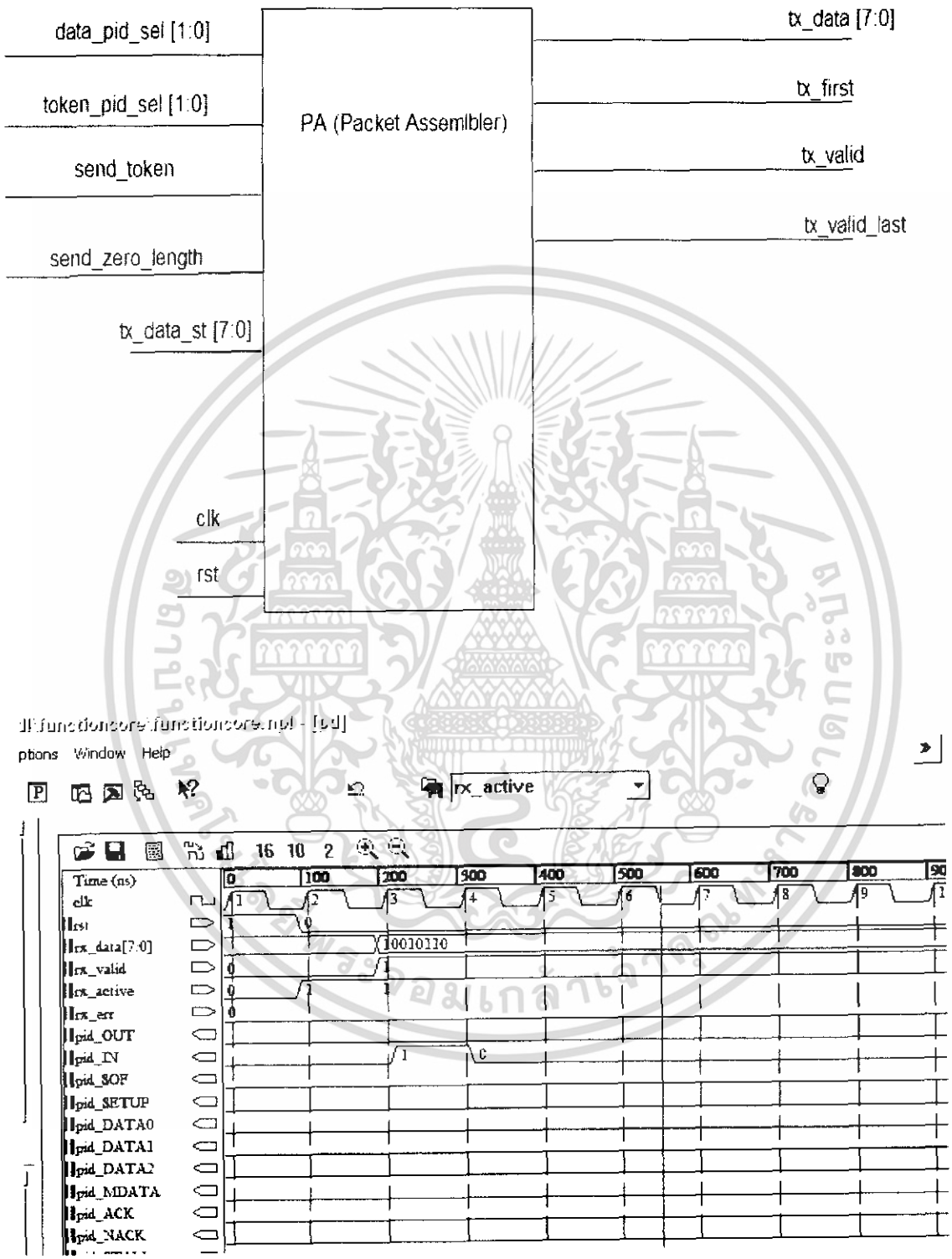
4.9.5 การออกแบบส่วนPacket Disassembly



รูปที่ 4.16 การออกแบบส่วนPacket Disassembly

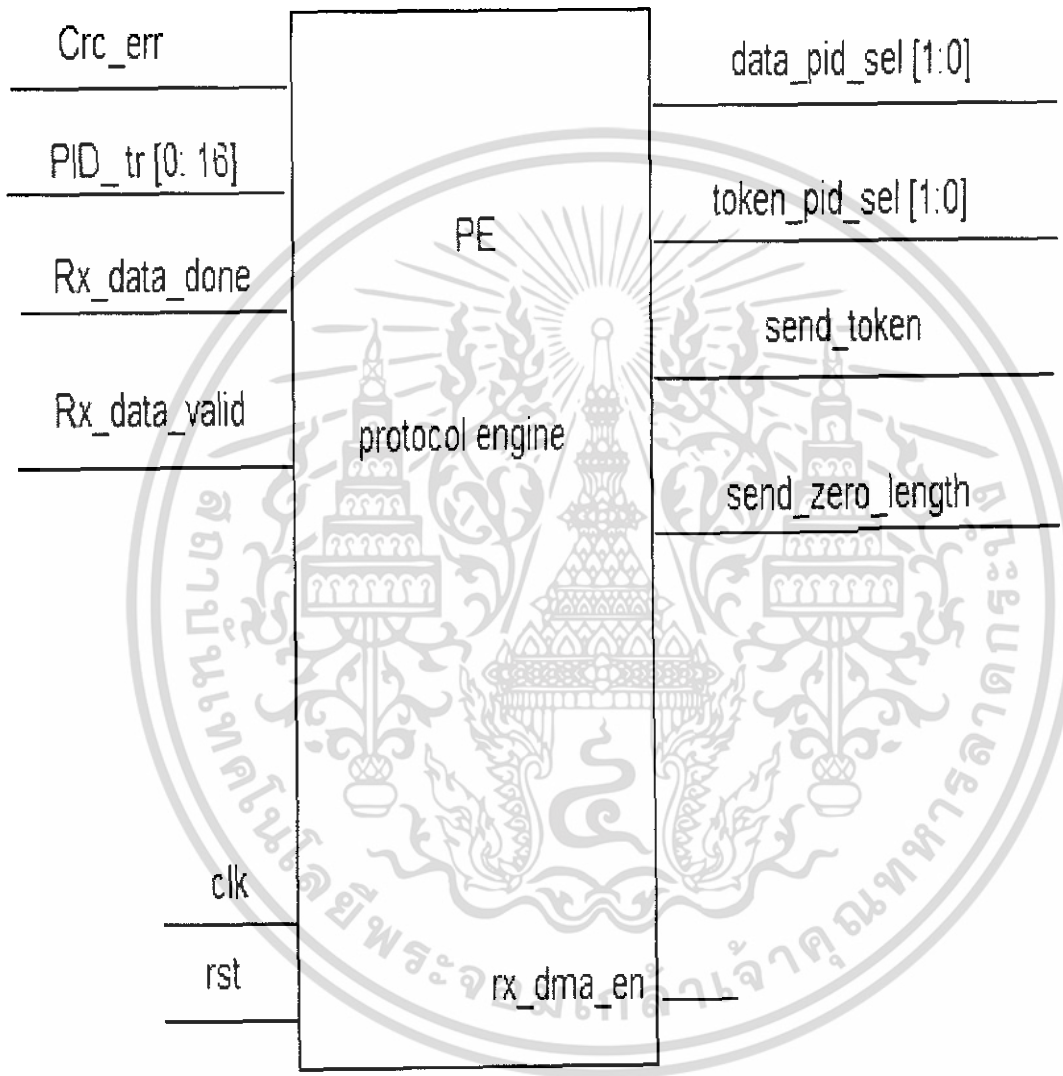


4.9.6 การออกแบบส่วนPacket Assembly



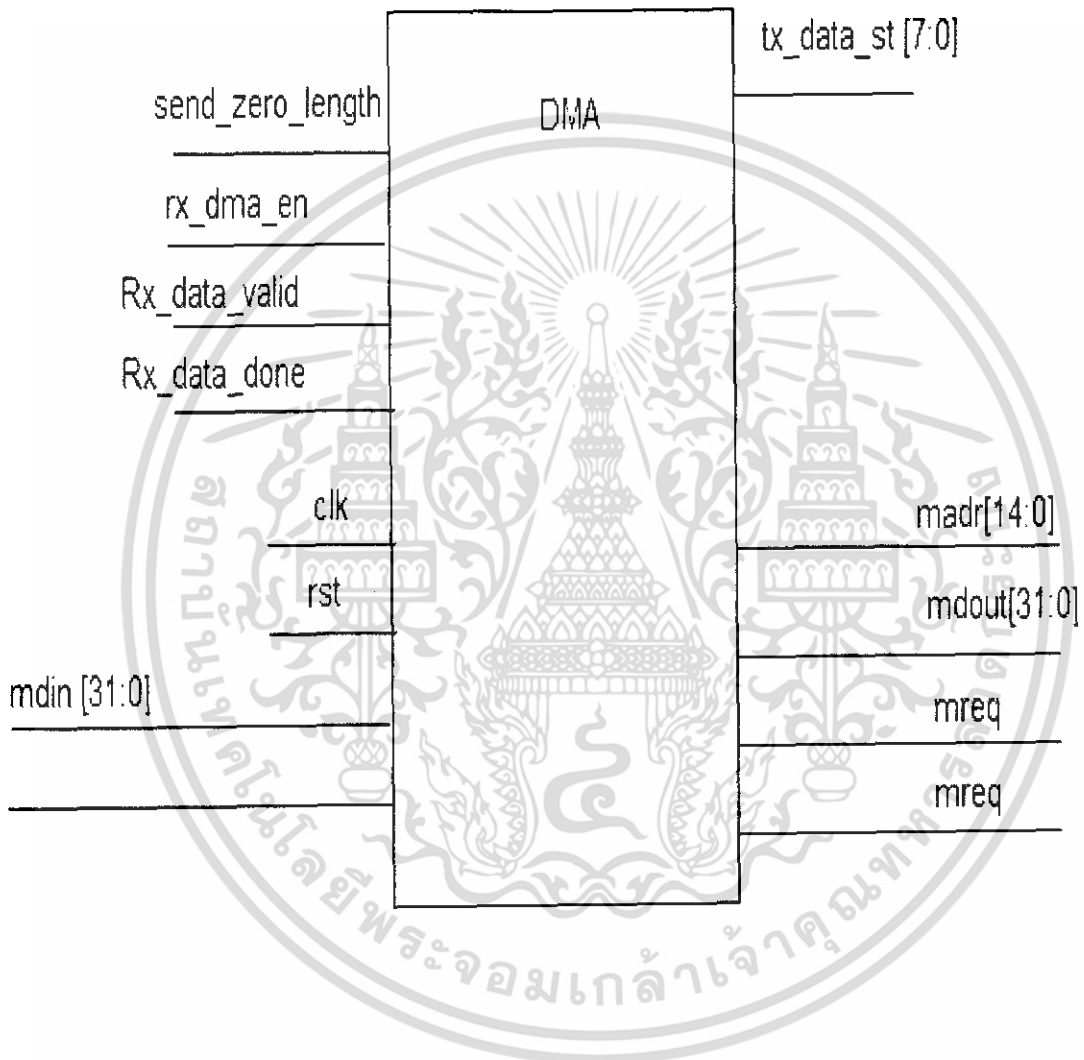
รูปที่ 4.17 การออกแบบส่วนPacket Assembly

4.9.7 การออกแบบส่วนProtocol Engine



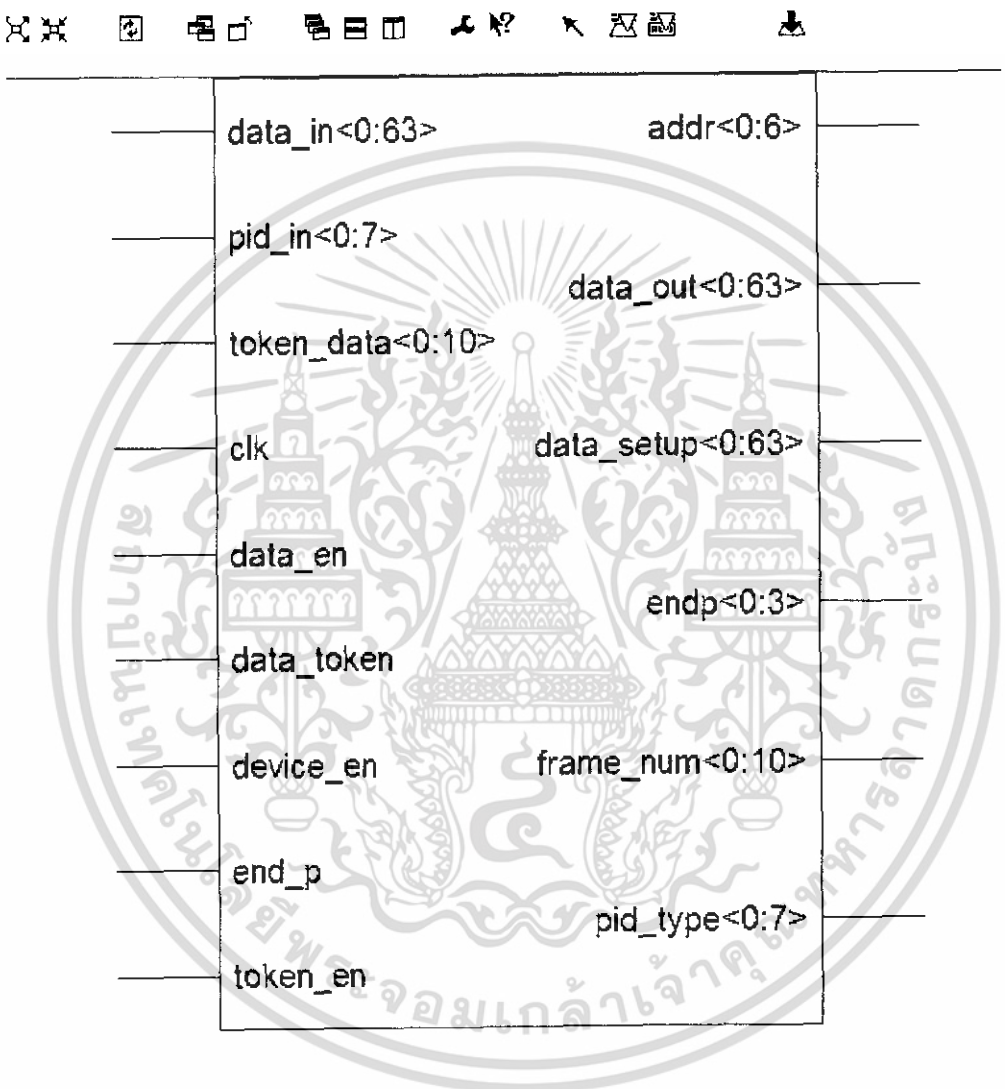
รูปที่ 4.18 การออกแบบส่วนProtocol Engine

4.9.8 การออกแบบในส่วนMemory Interface

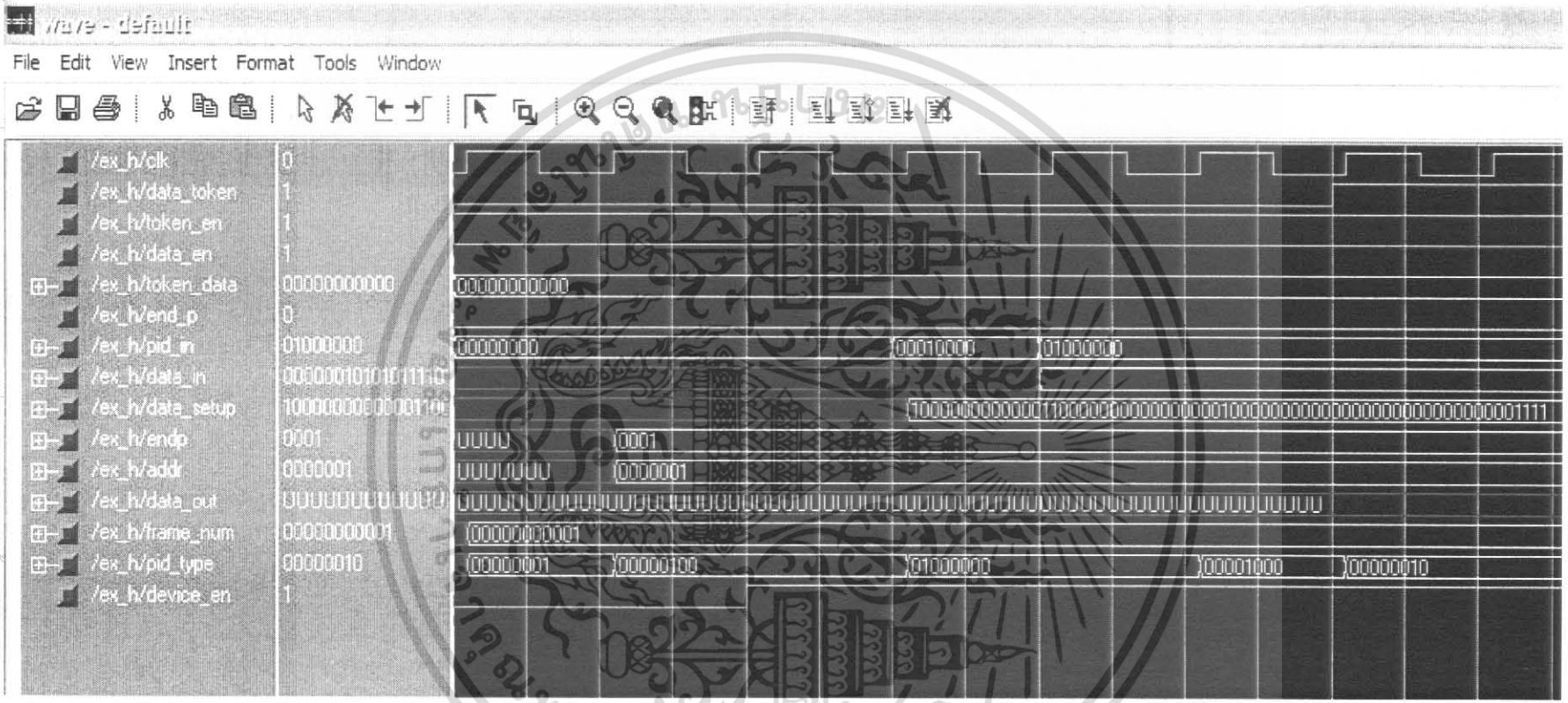


รูปที่ 4.19 การออกแบบในส่วนMemory Interface

4.9.9 การออกแบบในส่วนตัวโฮสคอนโทรลเลอร์



รูปที่ 4.20 การออกแบบในส่วนMemory Interface



รูปที่ 4.21 waveformการทำงานของโฮสคอนโทรลเลอร์ที่ออกแบบ

บทที่ 5
สรุปและวิจารณ์

5.1 บทวิเคราะห์ และ ความพึงพอใจกับผลงาน

-เนื่องจากเป็นสิ่งที่เป็นระบบภายในที่ต้องคิดออกแบบสร้างขึ้นเองโดยมีข้อจำกัดด้านข้อมูล และ เป็นแหล่งข้อมูลจากต่างประเทศ จึงทำให้การดำเนินงานในเทอมแรกค่อนข้างล่าช้า และอยู่ในรูปของทฤษฎีเป็นส่วนใหญ่

-ระบบที่ได้ทำการออกแบบขึ้น เป็นระบบที่เป็นไปตามที่มาตรฐานยูเอสเพียงในระดับหนึ่งเท่านั้น เนื่องจากในหลายๆส่วน ไม่สามารถหาข้อมูลเชิงลึกมาสนับสนุนและใช้ช่วยในการออกแบบให้สมบูรณ์ได้ ซึ่งยังมีอีกหลายๆส่วนที่จำเป็นต้องพัฒนาต่อไป ซึ่งปัจจัยที่จะทำให้การพัฒนาระบบให้สมบูรณ์ขึ้นภายในระยะเวลาจำกัด คือ การเพิ่มจำนวนบุคลากรร่วมทีมให้มากขึ้น เช่นเดียวกับที่ผู้ผลิตเพื่อการค้าที่ใช้ทีมงานในงานพัฒนาระบบจำนวนมาก

- ระบบที่ทำการออกแบบขึ้นรองรับการข้อผิดพลาดได้เพียงบางอย่างเท่านั้น ซึ่งในหลายๆข้อผิดพลาดที่อาจจะเกิดขึ้นจริง ยังไม่ได้ถูกคิดระบบให้ครอบคลุมไปถึง ทั้งนี้เนื่องจากระยะเวลาที่มีค่อนข้างจำกัดกว่าขอบข่ายของงานมาก ระบบภายในจริงเป็นแบบที่รองรับในกรณีปกติและเหตุการณ์สมมติบางเหตุการณ์ได้เท่านั้น ซึ่งสามารถพัฒนาให้ครอบคลุมกรณีผิดพลาดอื่นๆได้ต่อไป

5.2 สรุปผลการทดลองในภาคเรียนที่ 1

-บรรลุวัตถุประสงค์เรื่องการต้องการเพิ่มความรู้เรื่องUSB และนำมาออกแบบสร้างส่วนประกอบย่อยเบื้องต้นที่สามารถนำมาพัฒนาใช้งานจริงได้

-ได้โครงสร้างของระบบที่จะนำไปออกแบบสร้างต่อไปในเทอมที่เหลือ ซึ่งในส่วนที่ทำการพัฒนาสร้างจนเสร็จสิ้นแล้วคือ ส่วนตรวจสอบความถูกต้องของข้อมูลPacket ซึ่งให้ผลการทดลองที่ถูกต้อง น่าพึงพอใจ

-บรรลุเป้าหมายในการบุกเบิกสร้างระบบภายในของUSB ซึ่งสามารถใช้เป็นข้อมูลในการพัฒนาผลงานอื่นๆต่อไปได้

5.3 สรุปผลการทดลองในภาคเรียนที่2

- ผลการทดลองที่เหลือจากภาคเรียนแรกนั้น เป็นส่วนที่เกี่ยวกับการออกแบบระบบภายในล้วนๆ ซึ่งในระหว่างการออกแบบระบบนั้น มีปัญหาเกิดขึ้น ในหลายๆเรื่อง อาทิ เรื่องความผิดพลาดจากการใช้คำสั่ง เรื่องของการนำระบบย่อยๆมารวมกัน ซึ่งมักจะเป็นขั้นตอนที่มักเกิดข้อผิดพลาดได้ง่าย

- เนื่องจากความซับซ้อนของระบบภายในของระบบ USB ทำให้จำเป็นต้องลดขอบข่ายงานลงมาอยู่ที่การซิมมูเลทระบบทั้งหมดให้ทำงานได้โดยได้ยกเรื่อง การนำไปทดสอบต่อบนFPGA บอร์ด ไว้เป็นส่วนที่จะนำไปประยุกต์พัฒนาต่อได้ต่อไป ซึ่งจากผลการทดลองที่ได้ สามารถรับส่งข้อมูลกันในระหว่างการซิมมูเลชั่นได้ ซึ่งหากนำไปพัฒนาลงในFPGAต่อ ผู้ออกแบบจำต้องคำนึงถึงเรื่องของ TIMING เป็นสำคัญด้วย ซึ่งอาจมีผลนำไปสู่การปรับแก้ การเขียน VHDL ให้ได้ระบบที่มีประสิทธิภาพมากยิ่งขึ้นต่อไป

- ระบบที่ได้ทำการออกแบบขึ้นนั้นยังมีส่วนที่ต้องพัฒนาให้ดียิ่งขึ้นไปอีกในหลายๆส่วน ไม่ว่าจะเป็นส่วนของอัลกอลิธึ่มที่ใช้ในการสร้างระบบย่อยๆภายในแต่ละระบบ รวมไปถึง การออกแบบให้ตัวมินิโฮสคอนโทรลเลอร์สามารถทำการส่งถ่ายข้อมูลในโหมดการทำงานที่เหลืออื่นๆตามที่มาตรฐานUSBรองรับได้อีกด้วย

- เนื่องจากความไม่ชำนาญในการออกแบบระบบดิจิตอล ในเชิงพฤติกรรม ด้วย VHDL จึงทำให้ระบบต่างๆที่ทำการออกแบบในปริญญานิพนธ์เล่มนี้ ยังมีความไม่สมบูรณ์ ซึ่งผู้ที่จะนำไปศึกษาต่อควรเลือกใช้ในส่วนที่มีความสอดคล้องกับทางมาตรฐานเท่านั้น

เอกสารอ้างอิง

1. ลภน สุภาพ อรรถพล บุญยะโภคา วรพจน์ กรแก้ววัฒนกุล และ ชัยวัฒน์ ลิ้มพรจิตรวิไล. เรียนรู้และปฏิบัติการเชื่อมต่อคอมพิวเตอร์กับอุปกรณ์ภายนอกผ่าน พอร์ท USB ขั้นพื้นฐาน. INEX: กรุงเทพมหานคร.

2. วรเทพ ไพบูลย์รัตนากร และ บุญอนันต์ เกียงเอียง สัมผัสโลก USB ด้วย EZY USB Module. ASTRON LOGIC : กรุงเทพมหานคร 2543

3. Jan Axelson .USB complete: everything you need to develop custom USB peripherals. Madison, WI : Lakeview Research, c2001

4. John Hyde. USB design by example: a practical guide to building I/O devices. America: Inte Press, c2001

5. Universal Serial Bus Specification www.usb.org April 27, 2000

6. Rudolf Usselmann. USB Function IP Core www.asics.ws Rev. 1.5 , January 27, 2002

ภาคผนวก

CODE VHDL

Host Controller

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity host is
        port ( clk :in std_Logic ;
               data_token :in std_logic ;
               token_en   :in std_logic ;
               data_en    :in std_logic ;
               token_data :in std_logic_vector (0 to 10);
               end_p      :in std_logic ;
               pid_in     :in std_Logic_vector (0 to 7);
               data_in    :in std_logic_vector (0 to 63);

               data_setup :out std_logic_Vector (0 to 63);
               endp       :out std_logic_vector (0 to 3);
               addr       :out std_logic_vector (0 to 6);
               data_out   :out std_logic_vector (0 to 63);
               frame_num  :out std_logic_vector (0 to 10);
               pid_type   :out std_logic_Vector (0 to 7);
               device_en  :in std_logic      ) ;
end host;

architecture Behavioral of host is

signal data_device,data_real1 : std_logic_vector (0 to 63);
```

```
signal state :integer range 0 to 10 := 0;
begin
        process(clk )
        begin
                if rising_edge(clk) then

                        -- control setup
                        case state is
                                when    0 =>      -- send start frame
                                pid_type <= "00000001" ;
                                frame_num <= "00000000001"
;
                                state <= 1;
                                when  1 =>        -- and send setup
                                pid_type <= "00000100" ;
                                addr  <= "0000001" ;
                                endp      <= "0001" ;
                                state <= 2;

                                when  2 => --  send  data get port status
                                if pid_in ="00010000" then
                                        pid_type <=
"01000000" ;
                                        data_setup <=
"10000000" & "00000110" & "0000000000000001"  & "0000000000000000" &
"0000000000001111";
                                        state <= 3 ;
                                end if ;

                                when  3 => --     get data discritptor
                                if pid_in = "01000000"  then
```

```
                                        data_device <= data_in
;
                                        state <= 4;
                                    end if;
                                when  4 =>
                                    pid_type <= "00001000" ;
                                    state <= 5 ;
                                when 5 =>
                                    pid_type <= "00000010" ;
                                    state <= 6 ;
                                when 6 =>
                                    if pid_in = "01000000" then
                                        data_real1 <= data_in ;
                                        state <= 7 ;
                                    end if;
                                when 7 =>
                                    if data_en = '0' then
                                    state <= 0 ;
                                    end if;
                                when others =>
                                    endp <= "XXXX" ;
                            end case ;
                        end if;
                    end process;

end Behavioral;
```

Packet assembly

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity assembly is
        port    (addr           : in std_Logic_vector (0 to 6);
                 endp           : in std_logic_vector (0 to 3);
                 frame_num      : in std_logic_vector (0 to 10);

                 pid_type       : in std_logic_vector (0 to 7) ;
                 data_raw   : in std_logic_vector (0 to 15);
                 data_setup : in std_logic_vector (0 to 63);

                 data_ack   : out std_logic_vector (0 to 7);
                 data_token : out std_logic_vector (0 to 23);
                 data_real      : out std_logic_vector (0 to 50);
                 clk                    : in std_Logic  );

end assembly;

architecture Behavioral of assembly is
 signal din: std_logic_vector(10 downto 0);
 signal newcrc: std_logic_vector(4 downto 0);
 signal crc_in: std_logic_vector(4 downto 0):= "10101";
begin
                 process (clk)
```

```
begin
if (pid_type = "0000010") or (pid_type = "0000100") or (pid_type =
"0001000") then
                    -- crc5 for p_in ,p_out , p_setup
                    din <= addr & endp ;
elsif  pid_type= "0000001" then
                    din <= frame_num;
end if ;
                din <= "11000100100";
--------     CRC GENERATORS  --------
    newcrc(0) <= (((((((din(10) XOR din(9)) XOR din(6)) XOR din(5)) XOR din(3)) XOR
din(0)) XOR crc_in(0)) XOR crc_in(3)) XOR crc_in(4) ;
    newcrc(1) <= ((((((din(10) XOR din(7)) XOR din(6)) XOR din(4)) XOR din(1)) XOR
crc_in(0)) XOR crc_in(1)) XOR crc_in(4) ;
    newcrc(2) <= ((((((((((din(10) XOR din(9)) XOR din(8)) XOR din(7)) XOR din(6)) XOR
din(3)) XOR din(2)) XOR din(0)) XOR crc_in(0)) XOR crc_in(1)) XOR crc_in(2)) XOR crc_in(3))
XOR crc_in(4) ;
    newcrc(3) <= (((((((((din(10) XOR din(9)) XOR din(8)) XOR din(7)) XOR din(4)) XOR
din(3)) XOR din(1)) XOR crc_in(1)) XOR crc_in(2)) XOR crc_in(3)) XOR crc_in(4) ;
    newcrc(4) <= (((((((din(10) XOR din(9)) XOR din(8)) XOR din(5)) XOR din(4)) XOR
din(2)) XOR crc_in(2)) XOR crc_in(3)) XOR crc_in(4) ;
                end process;


process (clk)
begin
if rising_edge (clk) then


        case pid_type is
                        when "00000001" => -- sof
```

```
                                        data_token <=  "01011010" &
frame_num & newcrc;

                                    when "00000010" => -- packet in
                                        data_token <=  "00011000" &
addr & endp & newcrc;

                                    when "00000100" =>      -- packet out
                                        data_token <=  "10010110" &
addr & endp & newcrc;

                                    when "00001000" =>      -- setup
                                        data_token <=  "11010010" &
addr & endp & newcrc;

                                    when "00010000" =>      -- ack
                                        data_ack   <=  "00101101" ;

                                    when "00100000" =>      -- nak
                                        data_ack           <=
"10100101" ;

                                    when "01000000" =>  ---- data
                                        data_token <= "00111100" ;
                                    when others   =>
                                        null;
                            end case;


                        end if ;
```

```
                end process;

end Behavioral;
```

Packet Disassembly

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity de_assembly is
        port    (data_token : in std_logic_vector (0 to 23);
                 clk                    : in std_Logic ;
                 data_raw   : out std_logic_vector (0 to 15);
                 addr               : out std_Logic_vector (0 to 6);
                 endp               : out std_logic_vector (0 to 3);
                 frame_num          : out std_logic_vector (0 to 10);
                 pid_out            : out std_logic_vector (0 to 7));
end de_assembly;

architecture Behavioral of de_assembly is

begin
                 process (clk)
                 begin
```

```
        if rising_edge(clk) then
                case data_token(0 to 3) is
                when "0101" =>                  -- sof
                        pid_out <= "00000001" ;
                when "0001" =>                  -- packet in
                        pid_out <= "00000010" ;
                when "1001" =>                  -- packet out
                        pid_out <= "00000100" ;
                when "1101" =>                  -- setup
                        pid_out <= "00001000" ;
                when "0010" =>                  -- ack
                        pid_out <= "00010000" ;
                when "1010" =>                  -- nak
                        pid_out <= "00100000" ;
                when "0011" =>                  -- data
                        pid_out <= "01000000";
                when others =>
                                null;
                end case ;

                -- if pid is in, out , setup
                if (data_token(0 to 3) = "0001") or (data_token(0 to 3) =
"1001") or (data_token(0 to 3) = "1101") then
                        addr <= data_token (8 to 14);
                        endp <= data_token (15 to 18);
                -- if pid is start of frame
                elsif data_token (0 to 3) ="0101" then
                        frame_num <= data_token (8 to 18);
                end if ;
        end if;
    end process;

end Behavioral;
```