ระบบจำลองการควบคุมการเปิด-ปิดเครื่องใช้ไฟฟ้าภายในบ้านผ่านทางบลูทูธ

Home Electronic Device Control via Bluetooth

โดย

นายพิพัฒน์ เจริญวุฒิลาภ 44010334

นางสาววราภรณ์ สัจจะบุรุษ 44010422

นางสาวสาวิตรี จิตตะ 44010530

อาจารย์ที่ปรึกษา

รศ.ดร.กอบชัย เดชหาญ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ระบบจำลองการควบคุมการเปิด-ปิดเครื่องใช้ไฟฟ้าภายในบ้านผ่านทางบลูทูธ

Home Electronic Device Control via Bluetooth

โดย

นายพิพัฒน์ เจริญวุฒิลาภ

นางสาววราภรณ์ สัจจะบุรุษ

นางสาวสาวิตรี จิตตะ

ภาควิชา
วิศวกรรมโทรคมนาคม

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ปริญญานิพนธ์ปีการศึกษา 2547

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบจำลองการควบคุมการเปิด-ปิดเครื่องใช้ไฟฟ้าภายในบ้านผ่านทางบลูทูธ

Home Electronic Device control via Bluetooth

ผู้จัดทำ

1. นายพิพัฒน์ เจริญวุฒิลาภ 44010334
2. นางสาววราภรณ์ สัจจะบุรุษ 44010422
3. นางสาวสาวิตรี จิตตะ 44010530

..... อาจารย์ที่ปรึกษา

(รศ.ดร.กอบชัย เดชหาญ)

ระบบจำลองการควบคุมการเปิด-ปิดเครื่องใช้ไฟฟ้าภายในบ้านผ่านทางบลูทูธ

Home Electronic Device control via Bluetooth

โดย	นายพิพัฒน์	เจริญวุฒิลาภ	44010334
	นางสาววราภรณ์	สัจจะบุรุษ	44010422
	นางสาวสาวิตรี	จิตตะ	44010530

อาจารย์ที่ปรึกษา รศ.ดร.กอบชัย เดชหาญ

บทคัดย่อ

โครงงานนี้ได้ทำการสร้างระบบจำลองการเปิด-ปิดเครื่องใช้ไฟฟ้าภายในบ้านโดยทำการส่งสัญญาณควบคุมเครื่องใช้ไฟฟ้าผ่านทางเทคโนโลยีบลูทูธจากโทรศัพท์มือถือซึ่งจะทำให้ผู้ใช้สามารถควบคุมการทำงานของเครื่องใช้ไฟฟ้าภายในบ้านได้อย่างสะดวกและรวดเร็ว

Abstract

This project is to build the system to control the electronic device in the house. By using Bluetooth Technology in Mobile, user will get more convenient and easier to open or close the home electronic device.

สารบัญ

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญตาราง

หน้า

บทที่ 1

บทนำ

1.1. ความสำคัญและที่มาของปัญหา

ความสะดวกสบายนั้นเป็นสิ่งที่มนุษย์ทั่วไปต้องการ จึงทำให้มนุษย์ได้ทำการคิดประดิษฐ์สิ่งต่างๆ เพื่อมาอำนวยความสะดวกสบายตอบสนองกับความต้องการของตนเองไม่ว่าจะเป็นรถยนต์ที่อำนวยความสะดวกสบายในแง่ของการเดินทางไปยังที่ต่างๆ หรืออุปกรณ์อิเล็กทรอนิกส์ต่างๆที่มนุษย์ได้ใช้เวลาประดิษฐ์คิดค้นขึ้นมาเพื่อเป็นเครื่องอำนวยความสะดวกสบายให้กับตัวเอง เป็นต้น ในช่วงแรกของการพัฒนาอุปกรณ์อิเล็กทรอนิกส์นั้นอุปกรณ์ต่างๆจำเป็นที่จะต้องมีการต่อสายต่างๆ เพื่อไว้ใช้ในการควบคุมหรือแลกเปลี่ยนข้อมูลระหว่างอุปกรณ์จึงเป็นปัญหาในเรื่องของการจัดวางสาย จึงเป็นเหตุให้มีการคิดค้นและพัฒนาเทคโนโลยีไร้สายขึ้นมา ในปัจจุบันเทคโนโลยีไร้สายที่นิยมใช้กันอยู่กับอุปกรณ์ส่วนมากคือเทคโนโลยีไร้สายอินฟราเรด (Infrared Technology) ซึ่งมีข้อจำกัดในแง่ของทิศทางการให้สัญญาณกับอุปกรณ์ ด้วยเหตุนี้จึงมีความคิดที่จะเอาเทคโนโลยีไร้สายที่มีชื่อเรียกว่า เทคโนโลยีบลูทูธ (Bluetooth Technology) มาพัฒนาสร้างเป็นอุปกรณ์ไร้สาย โดยเทคโนโลยีนี้ใช้หลักการการส่งคลื่นวิทยุที่อยู่ในย่านความถี่ระหว่าง 2.4 – 2.5 GHz ข้อดีของเทคโนโลยีนี้คือมีขนาดเล็ก ราคาถูก และ มีความสามารถในการส่งสัญญาณได้ดีกว่าเทคโนโลยีไร้สายอินฟราเรด แต่ก็มีข้อจำกัดในเรื่องระยะของการส่งสัญญาณซึ่งระยะประมาณ 7-10 เมตร ดังนั้นจึงเหมาะสมกับการใช้เทคโนโลยีนี้ในบริเวณบ้านหรือในตัวอาคารที่มีบริเวณไม่กว้างนัก ในปัจจุบันนี้ผลิตภัณฑ์ต่างๆที่ใช้เทคโนโลยีไร้สายบลูทูธ (Bluetooth) ได้มีการผลิตออกมาใช้ในอุปกรณ์อิเล็กทรอนิกส์หลายๆชนิด เช่น โทรศัพท์เคลื่อนที่ คอมพิวเตอร์โน๊ตบุ๊ค คอมพิวเตอร์พ็อคเก็ตพีซี เป็นต้น แต่ก็ยังไม่มากพอกับอุปกรณ์ต่างๆที่มีใช้อยู่ในปัจจุบัน จึงได้มีความคิดที่จะพัฒนาฮาร์ดแวร์ที่ใช้เทคโนโลยีบลูทูธ เพื่อจะได้อุปกรณ์ไร้สายที่มีประสิทธิภาพมากขึ้นเนื่องจากเทคโนโลยีบลูทูธนั้นเป็นเทคโนโลยีที่มีมาได้ไม่นานนัก จึงเป็นปัญหาในด้านการขาดอุปกรณ์ที่จะนำมาพัฒนา เช่น ชิพบลูทูธ เป็นต้น อีกทั้งแหล่งข้อมูลต่างๆที่จะศึกษาหาข้อมูลการทำงาน การออกแบบฮาร์ดแวร์มีน้อย ดังนั้นปัญหาจึงอยู่ที่ว่า

- มีวิธีการอย่างไรที่จะหาฮาร์ดแวร์ของบลูทูธมาพัฒนาได้
- ทำอย่างไรจึงจะออกแบบฮาร์ดแวร์ได้
- จะนำเอาเทคโนโลยีบลูทูธมาพัฒนากับอุปกรณ์อะไร

1.2. วัตถุประสงค์ของโครงงานศึกษา

1. เพื่อศึกษาหลักการติดต่อสื่อสารของเทคโนโลยีไร้สายบลูทูธ
2. เพื่อศึกษาการออกแบบและสร้างอุปกรณ์ที่สามารถควบคุมโดยใช้เทคโนโลยีบลูทูธ
3. เพื่อศึกษาการเขียนโปรแกรมขึ้นมาติดต่อฮาร์ดแวร์
4. เพื่อพัฒนาอุปกรณ์ที่ใช้เทคโนโลยีบลูทูธเป็นอุปกรณ์ที่รู้จักและใช้อย่างกว้างขวาง
5. เพื่อลดต้นทุนการผลิตและขนาดเมื่อเทียบกับเทคโนโลยีไร้ที่ใช้อยู่ในปัจจุบัน

บทที่ 2

ทฤษฎีและหลักการ

ในปัจจุบันนี้ โลกของเรากำลังก้าวเข้าสู่ยุคดิจิตอล เนื่องจากสัญญาณดิจิตอลทำให้มนุษย์เราสามารถส่งถ่ายข้อมูลชนิดต่างๆจากที่หนึ่งไปยังอีกที่หนึ่งด้วยเวลาอันรวดเร็วเพียงพริบตา โดยยังสามารถคงความถูกต้องของข้อมูลไว้ได้ การส่งถ่ายข้อมูลดิจิตอลที่กล่าวมานั้น สามารถส่งผ่านตัวกลางได้หลายชนิด แต่ตัวกลางที่ใช้งานได้ง่ายและถูกที่สุดก็คือสายส่งสัญญาณที่มีตัวนำไฟฟ้าเป็นสื่อกลางหรือที่เราเรียกกันว่าสายไฟนั่นเอง การส่งข้อมูลผ่านสายไฟนั้นถูกนำมาใช้งานกันอย่างแพร่หลาย เนื่องจากความง่ายในการใช้งาน เพราะไม่ต้องเปลี่ยนสัญญาณข้อมูลไปอยู่ในรูปแบบอื่นก่อน อีกทั้งยังสามารถรองรับความเร็วในการส่งสัญญาณได้สูง แต่จากปัญหาที่ว่าการเชื่อมต่อสายไฟระหว่างอุปกรณ์ที่มากกว่าสองชิ้นเข้าด้วยกันทำให้เกิดความไม่สะดวกในการใช้งานอุปกรณ์ที่ไม่ได้ติดตั้งอยู่กับที่ จึงมีแนวความคิด ที่จะหาสื่อกลางชนิดอื่นเข้ามาทำหน้าที่แทนสายไฟที่ใช้กันอยู่แต่เดิม แต่เนื่องจากในปัจจุบัน ความต้องการข้อมูลรูปแบบต่างๆของมนุษย์มีเพิ่มสูงขึ้นเรื่อยๆ ทำให้ความเร็วในการส่งข้อมูลต้องเพิ่มสูงขึ้นเพื่อรองรับความต้องการที่ว่านี้ ดังนั้นสื่อกลางที่จะนำมาทดแทนนั้นก็ต้องสามารถรองรับความเร็วในการส่งข้อมูลที่สูงนี้ได้ด้วย

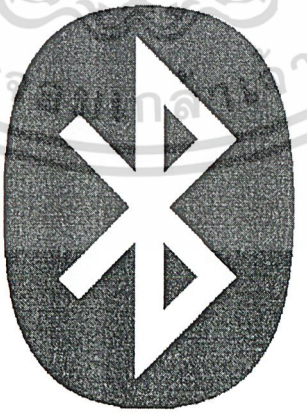
ตัวกลางที่ถูกนำมาทดแทนสายไฟที่นิยมใช้กันชนิดแรกก็คือแสงอินฟราเรด การส่งข้อมูลผ่านแสงอินฟราเรดถูกนำมาใช้งานอย่างแพร่หลาย เนื่องจากสามารถส่งข้อมูลด้วยความเร็วสูงได้ ประกอบกับความไม่ยุ่งยากของวงจรที่ใช้งาน แต่ข้อเสียของการใช้งานอินฟราเรด คือ ข้อจำกัดด้านระยะทางและตำแหน่งระหว่างอุปกรณ์ทั้งสองเนื่องจากการใช้แสงอินฟราเรดเป็นตัวกลางนั้นสามารถถูกรบกวนจากแสงภายนอกได้ง่าย ทำให้ระยะทางที่ใช้งานไม่สามารถเพิ่มได้มากนัก อีกทั้งตำแหน่งของอุปกรณ์รับส่งทั้งสองจะต้องอยู่ในแนวเส้นตรงเดียวกัน และ ไม่มีอะไรมาตัดขวางลำแสงจึงจะสามารถใช้งานได้ จากข้อเสียดังกล่าวทำให้เกิดความคิดที่จะนำคลื่นวิทยุมาใช้ส่งข้อมูลแทน เพราะคลื่นวิทยุสามารถรับส่งข้อมูลที่ความเร็วสูงได้ สามารถทะลุวัสดุต่างๆได้และไม่จำเป็นต้องให้ตัวส่งและรับอยู่ในแนวเส้นตรงเดียวกันขณะใช้งาน การนำคลื่นวิทยุมาใช้รับส่งข้อมูลสำหรับอุปกรณ์ระยะใกล้ ในปัจจุบันนี้มีกำหนดเป็นมาตรฐานอยู่ 3 ระบบด้วยกัน ระบบแรกคือ IEEE 802.11 หรือที่เรียกว่า Wireless LAN (WLAN) ออกแบบมาเพื่อสร้างระบบเน็ตเวิร์กไร้สายทดแทนการใช้สาย UTP หรือโคแอกเชียล ที่เชื่อมต่ออุปกรณ์ต่างๆ เข้ากับระบบ LAN แบบ อีเทอร์เน็ต (Ethernet) ระบบที่สองคือ HomeRF เป็นมาตรฐานที่กำหนดขึ้นเพื่อทดแทนการเชื่อมต่อของเครื่องใช้ไฟฟ้าที่อยู่ภายในบ้าน ซึ่งรูปแบบข้อมูลที่รับส่งนั้นจะมีทั้งภาพและเสียงรวมกันอยู่ ทำให้ต้องการความเร็วในการส่งข้อมูลที่สูงมาก และระบบสุดท้ายซึ่งเป็นระบบที่จะกล่าวถึงนี้คือ บลูทูธเป็นมาตรฐานที่กำหนดขึ้น เพื่อทดแทนการใช้สายเชื่อมต่อระหว่างอุปกรณ์ในระยะใกล้ อาทิเช่น การเชื่อมต่อระหว่างพรินเตอร์กับเครื่องคอมพิวเตอร์หรือระหว่างโทรศัพท์มือถือกับชุดหูฟังสมอลทอร์ก (small talk) เป็นต้น

เทคโนโลยีบลูทูธเป็นเทคโนโลยีสำหรับการเชื่อมต่ออุปกรณ์แบบไร้สายที่น่าจับตามองเป็นอย่างยิ่งในปัจจุบัน ทั้งในเรื่องความสะดวกในการใช้งานสำหรับผู้ใช้ทั่วไป และประสิทธิภาพในการทำงาน เนื่องจาก เทคโนโลยีบลูทูธ มีราคาถูก ใช้พลังงานน้อย และใช้เทคโนโลยีระยะใกล้ (Short Range) ซึ่งในอนาคตจะถูกนำมาใช้ในการพัฒนา เพื่อนำไปสู่การแทนที่อุปกรณ์ต่างๆที่ใช้สายเคเบิล เช่น Headset สำหรับโทรศัพท์เคลื่อนที่ เป็นต้น

2.1 จุดมุ่งหมายของบลูทูธ

บลูทูธเป็นมาตรฐานเปิด (Open Specification) ที่ออกแบบมาเพื่อใช้สำหรับส่งข้อมูลไร้สายระยะใกล้ (Short Range Wireless) ถูกกำหนดขึ้นโดย Bluetooth SIG (Bluetooth Special Interest Group) โดยมีจุดมุ่งหมายหลักในการในการวางมาตรฐานหลักๆ ดังนี้

- เป็นมาตรฐานเปิดที่ทุกคนสามารถใช้ข้อมูลที่ส่งผ่านบลูทูธนี้ได้ โดยไม่ต้องเสียค่าธรรมเนียมใดๆทั้งสิ้น เพื่อให้เกิดความแพร่หลายในการใช้งานและมีการพัฒนาระบบได้อย่างรวดเร็ว
- รับส่งข้อมูลไร้สายในระยะใกล้ คือ อุปกรณ์ต่างๆ สามารถส่งข้อมูลถึงกันได้โดยไม่ต้องใช้สาย
- สามารถรองรับการรับส่งเสียงและข้อมูลได้ในเวลาเดียวกัน นั่นคือระบบจะต้องมีความเร็วในการรับส่งข้อมูลเพียงพอสำหรับการส่งเสียงและข้อมูลไปพร้อมๆกัน
- สามารถใช้งานได้ในทุกที่ทั่วโลก หมายความว่าอุปกรณ์ที่ผลิตตามมาตรฐานของบลูทูธไม่ว่าจะผลิตจากผู้ผลิตใดหรืออยู่ ณ ตำแหน่งใดบนโลกสามารถใช้งานร่วมกันได้ จากจุดมุ่งหมายในข้อนี้ ทำให้ต้องใช้ความถี่คลื่นวิทยุที่สามารถใช้งานได้ในทุกประเทศ และมีการกำหนดสัญลักษณ์บลูทูธขึ้นดังรูปที่ 2.1



รูปที่ 2.1 สัญลักษณ์ของบลูทูธ

2.2 กำเนิดเทคโนโลยีบลูทูธ

ในปี 1994 เมื่อ Ericsson Mobile Communication ได้เริ่มศึกษาสิ่งที่จะมาทดแทนสายเคเบิลสำหรับอุปกรณ์ที่ใช้สำหรับโทรศัพท์เคลื่อนที่ โดยในการศึกษาจะมุ่งไปที่การใช้งานสัญญาณวิทยุ เนื่องจากสัญญาณวิทยุไม่จำเป็นจะต้องใช้การเดินทางแบบเส้นตรง ซึ่งเป็นข้อดีที่มีมากกว่าการเชื่อมต่อแบบอินฟราเรดที่ใช้ในการเชื่อมต่อระหว่างโทรศัพท์มือถือกับอุปกรณ์ในโทรศัพท์เคลื่อนที่รุ่นก่อนๆ และในการศึกษาไม่ได้มุ่งศึกษาเฉพาะการส่งข้อมูลเพียงอย่างเดียว ยังศึกษาถึงการส่งข้อมูลที่เป็นเสียงด้วย เพื่อใช้สำหรับ Headset ของโทรศัพท์เคลื่อนที่

2.3 The Bluetooth SIG

Bluetooth Special Interest Group (SIG) คือ กลุ่มของบริษัทที่ร่วมกันสนับสนุนเทคโนโลยีบลูทูธ เป็นกลุ่มที่วางแนวทางมาตรฐานของบลูทูธทั้งหมด ซึ่งถือกำเนิดขึ้นในปี 1998 โดยมีบริษัทยักษ์ใหญ่ 5 บริษัทเป็นสมาชิกยุคก่อตั้ง ได้แก่

1. Ericsson Mobile Communication AB.
2. Intel Corp.
3. IBM Corp.
4. Toshiba Corp.
5. Nokia Mobile Phones.

ในเดือน พฤษภาคม ปี 1998 บริษัทเหล่านี้ ได้ประกาศ การรวมตัวกัน เพื่อก่อตั้ง SIG และในเดือนกรกฎาคม ปี 1999 กลุ่มบริษัทเหล่านี้ได้ทำการเผยแพร่ Bluetooth Specification Version 1.0 ที่เว็บไซท์ http://www.bluetooth.com นอกจานี้ก็มีการเปิดรับสมาชิกของกลุ่มเพิ่มขึ้นเรื่อยๆ จนปัจจุบันนี้มีสมาชิกอยู่มากกว่า 1,800 บริษัท Bluetooth SIG ได้ทำการแบ่งกลุ่มพัฒนาออกเป็น 3 กลุ่ม โดยกลุ่มที่ 1 ทำการแก้ไขบลูทูธเวอร์ชัน 1.0 , กลุ่มที่ 2 ทำการพัฒนาเพิ่มเติมในเวอร์ชัน 1.0 และ กลุ่มที่ 3 ทำการพัฒนาบลูทูธเวอร์ชัน 2.0 โดยในเวอร์ชัน 2 ของบลูทูธนั้น ถูกคาดหวังให้มีอัตราการส่งข้อมูลสูงขึ้น (ระหว่าง 2 ถึง 10 Mb/s) รวมไปถึงการส่งข้อมูลที่เป็นมัลติมีเดีย (Multimedia) ซึ่งจะเป็นที่แพร่หลายในอนาคต

ในส่วนของชื่อบลูทูธนั้นได้มาจากชื่อของ Harald Blatant กษัตริย์ของเดนมาร์ก ซึ่งในช่วงที่ครองราชย์อยู่นั้นสามารถรวมเดนมาร์กและนอร์เวย์เข้าด้วยกัน เปรียบเสมือนเทคโนโลยีบลูทูธที่จะรวมอุตสาหกรรมด้านการสื่อสารและคอมพิวเตอร์เข้าด้วยกัน จากชื่อ Blatant เมื่อพิจารณาการออกเสียงจากภาษาดั้งเดิมโดยผิดเพี้ยนน้อยที่สุดก็จะได้ชื่อ Bluetooth และสาเหตุที่เลือกใช้ชื่อนี้ก็เพราะว่า อริคสัน ซึ่งเป็นผู้เริ่มเทคโนโลยีมีบริษัทแม่อยู่ที่เดนมาร์กนั่นเอง

2.4 วัตถุประสงค์

สาเหตุที่เทคโนโลยีบลูทูธเป็นที่สนใจสำหรับกลุ่มบริษัทต่างๆเนื่องจาก เทคโนโลยีนี้ทำให้การเชื่อมต่อสำหรับโทรศัพท์เคลื่อนที่กับอุปกรณ์ต่างๆทำได้สะดวกขึ้น และบริษัทต่างๆสามารถทำกำไรจากเทคโนโลยีนี้ได้ โดยการขายผลิตภัณฑ์ที่ใช้เทคโนโลยีบลูทูธนี้ รวมไปถึงการขายซอฟท์แวร์ที่ใช้สำหรับการใช้งานอุปกรณ์ที่บริษัทได้ผลิตขึ้นมา จุดประสงค์ของเทคโนโลยีบลูทูธนั้นเริ่มต้นเพื่อขายให้แก่บริษัทผู้ผลิตโทรศัพท์เคลื่อนที่ เนื่องจากเทคโนโลยีนี้สามารถทำให้ผลิตภัณฑ์ของกลุ่มบริษัทเหล่านี้ใช้งานได้ดีขึ้น โดยการเพิ่มขีดความสามารถของการติดต่อสื่อสารระหว่างโทรศัพท์เคลื่อนที่กับอุปกรณ์ต่างๆ เนื่องจากในอดีตการสื่อสารนี้ทำได้โดยใช้สายเคเบิล ซึ่งมีความเสี่ยงที่ข้อมูลจะเกิดความเสียหายได้ ดังนั้นจุดประสงค์ของเทคโนโลยีบลูทูธ คือ การแทนที่การสื่อสารระหว่างอุปกรณ์เคลื่อนที่ต่างๆ ที่ใช้สายเคเบิลมาเป็นใช้อุปกรณ์ที่ใช้เทคโนโลยีบลูทูธ เนื่องจากเทคโนโลยีบลูทูธถูกออกแบบมาให้ใช้กับอุปกรณ์เคลื่อนที่ต่างๆ เช่น โทรศัพท์เคลื่อนที่ จึงสามารถทำงานได้โดยใช้แบตเตอรี่ เทคโนโลยีบลูทูธเป็นเทคโนโลยีที่ใช้พลังงานน้อยและสามารถทำงานได้แม้ขณะที่มีแรงดันไฟฟ้าต่ำ ดังนั้นเทคโนโลยีบลูทูธจึงถูกนำไปใช้กับอุปกรณ์ขนาดเล็กอื่นๆ เช่น Headset และ PDAs ในปัจจุบันการแทนที่สายเคเบิลด้วยเทคโนโลยีบลูทูธยังมีปัญหาอยู่ในบางเรื่อง เช่น ราคาของอุปกรณ์ที่ใช้เทคโนโลยีบลูทูธนั้นมีราคาสูงกว่าแบบใช้สายเคเบิลอยู่มาก ดังนั้นถ้าอุปกรณ์ที่ใช้เทคโนโลยีบลูทูธนี้สามารถบรรลุวัตถุประสงค์เหล่านี้ได้ คือ ราคาไม่แพง, ใช้งานง่าย, มีเสถียรภาพสูง, ขนาดเล็กและใช้พลังงานต่ำ เทคโนโลยีบลูทูธจะถูกนำมาใช้แทนที่การติดต่อสื่อสารแบบที่ใช้สายได้อย่างแพร่หลายแน่นอน

2.5 Protocol Stack

สำหรับอุปกรณ์ที่มีการเชื่อมต่อเข้าด้วยกันเป็นเน็ตเวิร์กนั้น การส่งข้อมูลจากอุปกรณ์ต้นทางไปยังอุปกรณ์ปลายทาง จำเป็นต้องมีการส่งข้อมูลอื่นๆประกอบเข้าไปกับข้อมูลที่ต้องการส่งนั้นด้วย เพื่อควบคุมเส้นทางของข้อมูลให้สามารถส่งไปถึงอุปกรณ์ปลายทางได้อย่างถูกต้อง ทำให้การส่งข้อมูลแต่ละครั้งเกิดการทำงานต่างๆขึ้นมากมาย จึงเกิดการสร้างโมเดลแทนการทำงานต่างๆที่ว่านี้ขึ้น เพื่อให้สามารถมองเห็นภาพรวมของการทำงานทั้งหมดได้

โปรโตคอลสแต็ก (Protocol stack) เป็นส่วนโครงสร้างการทำงานของโปรแกรมที่เข้ามาควบคุมการทำงานของอุปกรณ์บลูทูธหรือที่มักเรียกกันว่า “ไดร์ฟเวอร์” (dirver) ซึ่งจะเป็นตัวที่อนุญาตให้แอพพลิเคชันซอฟท์แวร์ (application software) ส่งและรับข้อมูลจากตัวอุปกรณ์บลูทูธได้

สำหรับโมเดลการทำงานของบลูทูธ (Bluetooth Module) ถูกกำหนดให้มีโครงสร้างการทำงานดังตารางที่ 2.1 ซึ่งจะเห็นได้ว่ามีจำนวน 8 ชั้น มากกว่าโมเดล OSI (OSI Model) อยู่ 1 ชั้น ทำให้ขอบเขตการทำงานในแต่ละชั้นแตกต่างจากโมเดล OSI แต่ลำดับการทำงานมีลักษณะเหมือนกัน

Application Layer	Applications
Presentation Layer	RFCOMM/SDP
Session Layer	L2CAP
Transport Layer	HCI
Network Layer	Link Manager
Data Link Layer	Link Controller
Physical Layer	Base band
	Radio
OSI Model	**Bluetooth Module**

ตารางที่ 2.1 การเปรียบเทียบ OSI Model กับ Bluetooth Module

การทำงานของโมเดลบลูทูธในแต่ละชั้นมีหน้าที่หลักดังนี้

- ชั้นที่ 1 Radio เป็นส่วนที่เกิดการรับและส่งคลื่นวิทยุจริงๆ เป็นส่วนของวงจรฮาร์ดแวร์ภาคส่งและรับคลื่นวิทยุที่ถูกควบคุมจากชั้น Base band ไม่ว่าจะเป็นความถี่และระดับความแรงของสัญญาณที่ใช้ รวมไปถึงเฟรมข้อมูลที่จะส่ง

- ชั้นที่ 2 Base band การทำงานของชั้นนี้ถือได้ว่าเป็นหัวใจของ Bluetooth ในด้านฮาร์ดแวร์เลยก็ว่าได้ หน้าที่หลักของชั้นนี้คือ การควบคุมวงจรภาคส่งและรับคลื่นวิทยุที่อยู่ชั้นล่างสุด ซึ่งจุดสำคัญที่สุดของการควบคุมก็คือ การเลือกช่องความถี่ในการรับส่งข้อมูลให้ตรงกันระหว่าง Master และ Slave ที่ต้องมีการกระโดดไปในรูปแบบเดียวกัน

- ชั้นที่ 3 Link Controller ควบคุมการเชื่อมต่อพื้นฐานของบลูทูธทั้งหมด ไม่ว่าจะเป็นสถานะของอุปกรณ์, โหมดการทำงานของอุปกรณ์, การค้นหาอุปกรณ์ Bluetooth ใกล้เคียง รวมไปจนถึงการเลือกว่าจะเป็น Master หรือ Slave ในสภาพแวดล้อมต่างๆ

- ชั้นที่ 4 Link Manager โดยรวมแล้วทำหน้าที่แปลงคำสั่งที่ได้รับจากชั้นบนเป็นลำดับหน้าที่การทำงานที่ชั้นล่างรู้จัก และคอยส่งคำสั่งลงไปควบคุมการทำงานของชั้นล่างทั้งหมด Link Manager เป็นส่วนที่ควบคุมจัดการ link ต่าง ๆ ได้แก่ link setup, link configuration, link packet control and transfer ตลอดจนจัดการ link security ในระหว่างช่วงกำหนดค่าตั้งต้นของการเชื่อมต่อรวมถึงขณะที่ยังคงเชื่อมต่ออยู่ด้วย อีกทั้งเป็นตัวควบคุมหมวดการทำงานแบบประหยัดกำลังไฟฟ้า (power saving mode)

- ชั้นที่ 5 HCI (Host Control Interface) เป็นส่วนที่จัดเตรียมอินเตอร์เฟสมาตรฐาน (standrard interface) ระหว่างอุปกรณ์บลูทูธ (Bluetooth module) และ Link Manager เป็นโปรโตคอลเชื่อมต่อระหว่างโปรแกรมชั้นบนที่ทำงานอยู่บนระบบหนึ่ง (เช่นโปรแกรมในเครื่องคอมพิวเตอร์โน้ตบุ๊กทำงานบน

CPU x86) กับส่วนควบคุมการทำงานของบลูทูธ (เช่น การ์ด PCMCIA Bluetooth ที่ต่ออยู่ในเครื่องคอมพิวเตอร์โน้ตบุ๊ก) ทำให้โปรแกรมรู้จักคำสั่งควบคุมอุปกรณ์บลูทูธ

- ชั้นที่ 6 L2CAP (Logical Link Control and Adaptation Protocol) ประกอบด้วย protocol multiplexing, segmentation and reassembly, quality of service โครงสร้างของ L2CAP จะจัดการเกี่ยวกับช่องสัญญาณ (channel) เมื่อมีการเชื่อมต่อกัน โดยจะกำหนดให้ทุกช่องสามารถมีการติดต่อแบบ full-duplex คือ ติดต่อได้ทั้งรับและส่งในทุกช่อง และยังทำหน้าที่หน้าที่มัลติเพล็กซ์ข้อมูลจากชั้นบนซึ่งอาจจะมีการทำงานของโปรแกรมหลายโปรแกรมพร้อมกัน และจัดแบ่งข้อมูลออกเป็นแพ็กเก็ต ทั้งยังเป็นตัวทำให้แอพพลิเคชัน (application) สามารถใช้งานโปรโตคอลระดับสูงบางตัวได้ (higher-layer protocol) เช่น TCP/IP, RFCOMM

- ชั้นที่ 7 RFCOMM/SDP สำหรับ RFCOMM เป็นโปรโตคอลเสมือน ที่ทำให้แอพพลิเคชันด้านบน มอง บลูทูธ เป็นเหมือนพอร์ตอนุกรม (Serial Port) ทั่วไป ส่วน SDP (Service Discovery Protocol) เป็นโปรโตคอลที่ช่วยค้นหาบริการจากอุปกรณ์บลูทูธตัวอื่นที่อยู่ในขอบเขตพิโกเน็ตเดียวกัน SDP เป็นเลเยอร์ (layer) ที่เปิดทางให้กับบริการระดับสูง (high-level services) ตัวอย่างเช่น LAN access หรือ printer ให้ผู้ใช้และแอพพลิเคชันอื่น ๆ เข้าถึงกันได้

- ชั้นที่ 8 Applications เป็นส่วนของโปรแกรมที่ติดต่อรับหรือส่งข้อมูลกับผู้ใช้

2.6 การติดต่อระหว่างอุปกรณ์บลูทูธ (Bluetooth Connection)

การติดต่อถึงกันระหว่างอุปกรณ์บลูทูธนั้น จะกระทำได้ภายในรัศมีการส่งสัญญาณวิทยุคลื่นสั้นความถี่ประมาณ 2.4 GHz กล่าวคือ ต้องอยู่ภายในรัศมี 7 - 10 เมตร จึงจะสามารถติดต่อถึงกันได้ ซึ่งการติดต่อหรือเชื่อมต่อถึงกันของอุปกรณ์บลูทูธนั้นมีด้วยกัน 2 แบบ คือ

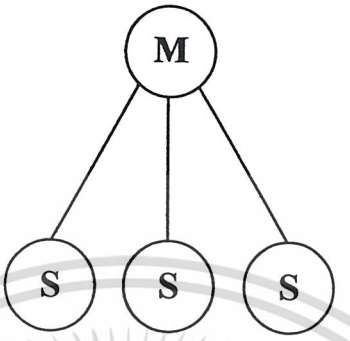
2.6.1 การเชื่อมต่อแบบพอยนท์-ทู-พอยนท์ (point-to-point)

เป็นการเชื่อมต่อระหว่างอุปกรณ์บลูทูธในลักษณะตัวต่อตัว โดยอุปกรณ์ตัวหนึ่งทำหน้าที่เป็นมาสเตอร์ (master) และอีกตัวหนึ่งทำหน้าที่เป็นสเลฟ (slave) ดังรูปที่ 2.2

รูปที่ 2.2 การเชื่อมต่อแบบพอยนท์-ทู-พอยนท์ (point-to-point)

2.6.2 การเชื่อมต่อแบบพอยนท์-ทู-มัลติพอยนท์ (point-to-multipoint)

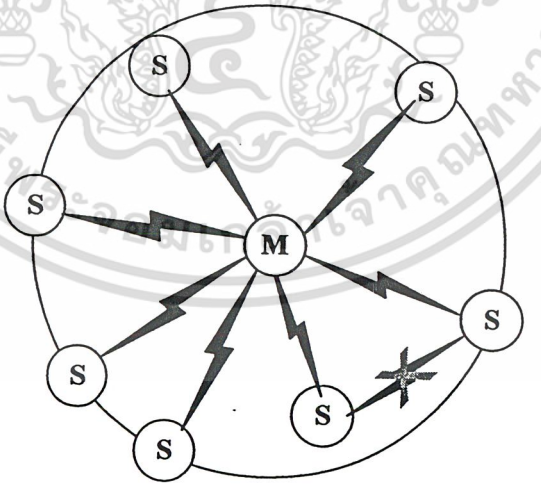
เป็นการเชื่อมระหว่างอุปกรณ์บลูทูธตัวหนึ่งที่ทำหน้าที่เป็นมาสเตอร์(master)กับอุปกรณ์บลูทูธที่ทำหน้าที่เป็นสเลฟ (slave) หลาย ๆ ตัวในเวลาเดียวกัน ดังรูปที่ 2.3



รูปที่ 2.3 การเชื่อมต่อแบบพอยนท์-ทู-มัลติพอยนท์ (point-to-multipoint)

2.7 รูปแบบเครือข่ายของบลูทูธ (Bluetooth Network Topology)

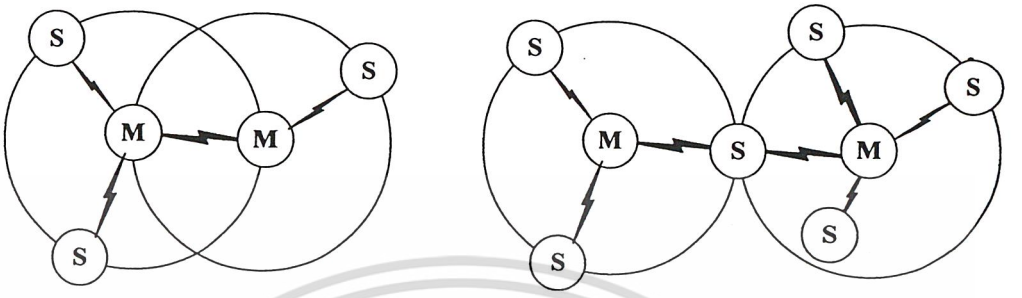
ระบบเครือข่าย (network) ระดับแรกสุดของบลูทูธ มีชื่อเรียกว่า "พิโคเน็ท" (piconets) ซึ่งใน 1 วงเครือข่ายพิโคเน็ทนั้นจะประกอบด้วยอุปกรณ์บลูทูธไม่เกิน 8 ตัว โดยจะมีอุปกรณ์ตัวหนึ่งทำหน้าที่เป็นมาสเตอร์ (master) ส่วนอุปกรณ์ที่เหลือจะทำหน้าที่เป็นสเลฟ (slave) ซึ่งสเลฟนั้นจะไม่สามารถเชื่อมต่อถึงกันได้โดยตรง แต่จะสามารถติดต่อถึงกันได้ผ่านทางมาสเตอร์ ดังแสดงในรูปที่ 2.4



รูปที่ 2.4 วงพิโคเน็ท (piconet) ที่มีการเชื่อมต่อระหว่างอุปกรณ์บลูทูธ 8 ตัว

การที่มีวงพิโคเน็ทตั้งแต่ 1 วงขึ้นไปแต่ไม่เกิน 10 วงมาเชื่อมต่อถึงกัน โดยมีการใช้อุปกรณ์บลูทูธในแต่ละวงร่วมกัน (share device) ทำให้เกิดระบบเครือข่ายขึ้นอีกระดับหนึ่ง ซึ่งเรียกว่า "สแกทเทอร์เน็ท" (scatternet) ส่งผลให้รัศมีการส่งสัญญาณเพิ่มขึ้น เนื่องจากสามารถส่งสัญญาณจากวงพิโคเน็ทหนึ่ง

ไปยังอีกวงหนึ่งได้ แต่ในขณะเดียวกันก็ส่งผลให้การส่งสัญญาณช้าลงหากต้องส่งไปยังวงที่อยู่ไกลมาก รูปที่ 2.5 เป็นตัวอย่างของเครือข่ายสแกทเทอร์เน็ท



รูปที่ 2.5 วงพิโคเน็ท (piconet) 2 วงมาเชื่อมต่อกันเป็นสแกทเทอร์เน็ท (scatternet)

สแกทเทอร์เน็ทจะมีจำนวนวงของพิโคเน็ทได้มากที่สุดเพียง 10 วงเท่านั้น ดังนั้น ขอบเขตสูงสุดของการติดต่อจากอุปกรณ์บลูทูธตัวหนึ่งไปยังอีกตัวหนึ่ง คือ ประมาณ 100 เมตร เนื่องจากใน 1 วงพิโคเน็ทมีรัศมีประมาณ 10 เมตร

2.8 ระบบรักษาความปลอดภัยของบลูทูธ (Bluetooth Security)

ในการส่งสัญญาณระหว่างอุปกรณ์บลูทูธนั้นมีระบบรักษาความปลอดภัย (security) เพื่อป้องกันการรุกล้ำเข้ามาล้วงเอาข้อมูล (hack) โดยแบ่งการทำงานออกเป็น 3 หมวด (mode) ดังนี้

2.8.1 ไม่ใช้ระบบรักษาความปลอดภัย (Unsecured)

ไม่มีการป้องกันการล้วงเอาข้อมูล โดยผู้ใช้ที่มีอุปกรณ์บลูทูธสามารถเข้ามาเชื่อมต่อถึงกันในเครือข่ายได้ทันทีเมื่อเข้ามาในขอบเขตรัศมีการติดต่อ

2.8.2 ระบบรักษาความปลอดภัยแบบเซอร์วิส (Service secure)

มีการแสดงตัวผู้ใช้ (user) ก่อนการเข้าถึงหรือเชื่อมต่อระหว่างกัน เพื่อเป็นการยืนยันว่าผู้ที่จะเข้ามาเชื่อมต่อเป็นผู้ที่ได้รับการอนุญาตให้เข้ามาเชื่อมต่อได้

2.8.3 ระบบรักษาความปลอดภัยแบบลิงค์ (Link secure)

มีการแสดงตัวผู้ใช้เช่นเดียวกันกับระบบรักษาความปลอดภัยแบบเซอร์วิส (Service secure) แต่เพิ่มความปลอดภัยมากขึ้นด้วยการเข้ารหัส (Encryption) ข้อมูลที่จะทำการส่งถึงกันก่อนที่จะทำการส่งสัญญาณ ซึ่งหากมีการล้วง (hack) เอาข้อมูลระหว่างที่ทำการส่งสัญญาณนั้น ผู้ล้วงเอาข้อมูล (hacker) ก็จะไม่สามารถเข้าใจข้อมูลที่ล้วงเอาไปนั้นได้ เนื่องจากได้ทำการเข้ารหัสเอาไว้ก่อนแล้ว

2.9 ช่วงความถี่ที่ใช้งาน

หลักการพื้นฐานสำหรับการส่งข้อมูลผ่านคลื่นวิทยุก็คือการมอดูเลท (Modulate) หรือถ้าพูดอย่างง่ายๆก็คือ การแฝงเอาข้อมูลรวมเข้าไปไว้กับคลื่นวิทยุแล้วส่งออกอากาศไป เมื่อไปถึงปลายทางตัวรับก็จะแยกเอาตัวข้อมูลออกมาจากคลื่นวิทยุแล้วนำข้อมูลที่ได้ออกไปใช้งาน คลื่นวิทยุที่ทำหน้าที่เป็นตัวพาข้อมูลไปยังปลายทางนี้เราเรียกว่าคลื่นพาหะ ส่วนกระบวนการในการมอดูเลทข้อมูลเข้าไปในคลื่นพาหะนั้นก็มีอยู่หลายวิธี ซึ่งในแต่ละวิธีก็จะมีข้อดีข้อเสียและความเหมาะสมกับงานแต่ละอย่างแตกต่างกันกันไป

จากเป้าหมายหลักข้อหนึ่งของบลูทูธ ที่จะทำให้สามารถใช้งานได้ในทุกๆประเทศ ความถี่ที่จะใช้เป็นคลื่นพาหะจึงจำเป็นต้องอยู่ในช่วงที่ทั่วโลกเปิดให้ใช้งานได้โดยไม่ต้องขออนุญาต ช่วงความถี่ดังกล่าวคือ ช่วงความถี่ ISM (Industrial, Scientific and Medical Band : ISM Band) ซึ่งมีความถี่ที่ใช้งานอยู่ในช่วง 2.4 GHz เนื่องจากช่วงความถี่นี้เป็นช่วงความถี่ที่สงวนไว้ใช้งานทางด้านอุตสาหกรรม วิทยาศาสตร์และการแพทย์ที่ทุกประเทศทั่วโลกเปิดให้ใช้งานได้

ช่วงความถี่ ISM นั้นถูกกำหนดขอบเขตไว้ในช่วง 2,400 ถึง 2,483.5 MHz โดยมี Lower Guard Band (LGB) เท่ากับ 2 MHz และ Upper Guard Band (UGB) เท่ากับ 3.5 MHz และจะถูกแบ่งเป็นช่องๆ เพื่อให้อุปกรณ์หลายๆ ตัวทำงานพร้อมกันได้ตามหลักของการทำ Frequency Division Multiplex (FDM) โดยแต่ละช่องจะมีความกว้างหรือแบนวิดธ์เท่ากับ 1 MHz

จากข้อกำหนดดังกล่าวจะได้ความถี่ใช้งานเริ่มต้นที่ 2,402 MHz (2,400 + LGB) และความถี่ที่ใช้งานสุดท้ายอยู่ที่ 2,480 MHz (2,483.5 – UGB) เมื่อให้ความกว้างของแต่ละช่องสัญญาณเท่ากับ 1 MHz จะได้จำนวนช่องสัญญาณทั้งหมดเท่ากับ 79 ช่อง แต่เนื่องจากยังมีบางประเทศ เช่น ฝรั่งเศส ที่ไม่เปิดช่วงความถี่ ISM เต็มช่วง (โดยเหลือช่องสัญญาณให้ใช้งานได้เพียง 23 ช่อง) แต่คาดว่าจะมีการอนุญาตให้ใช้งานได้เต็มช่วงในอนาคต ดังแสดงในตารางที่ 2.2

ประเทศ	ช่วงความถี่ ISM (GHz)	Lower Guard Band (LGB)	Upper Guard Band (UGB)	จำนวนช่องสัญญาณที่ใช้
ฝรั่งเศส	2.4465-2.4835	7.5 MHz	7.5 MHz	23
ประเทศอื่นๆ	2.4000-2.4835	2 MHz	3.5 MHz	79

ตารางที่ 2.2 ความถี่และช่องสัญญาณ ISM Band

นอกจากข้อกำหนดด้านความถี่ที่จะทำให้อุปกรณ์ต่างๆ สามารถใช้งานได้ทั่วโลกแล้ว ยังมีข้อจำกัดอีกส่วนหนึ่งที่แต่ละประเทศมีการตั้งข้อกำหนดไว้ นั่นคือ เรื่องของกำลังส่งของอุปกรณ์ ซึ่งจะส่งผลโดยตรงต่อระยะห่างไกลสุดที่สามารถเชื่อมต่อกันได้ อย่างที่เข้าใจกันง่ายๆ ว่า ถ้ากำลังส่งแรงระยะห่างสูงสุดระหว่างอุปกรณ์ที่ส่งได้ก็จะมากตาม แต่เนื่องจากบลูทูธถูกออกแบบมาสำหรับการส่ง

ข้อมูลในระยะใกล้ ทำให้อุปกรณ์ไม่ต้องใช้กำลังส่งที่สูงมาก ซึ่งประเทศต่างๆ ให้ใช้งานได้โดยไม่ต้องขออนุญาต

คลาส	กำลังส่งสูงสุด	กำลังส่งต่ำสุด	ระยะการใช้งาน (โดยประมาณ)
1	100 mW (20 dBm)	1 mW (0 dBm)	100 เมตร
2	2.5 mW (4 dBm)	0.25 mW (-6 dBm)	10 - 20 เมตร
3	1 mW (0 dBm)	-	5 – 10 เมตร

ตารางที่ 2.3 การแบ่งคลาสตามอุปกรณ์ของกำลังส่ง

ตารางที่ 2.3 แสดงให้เห็นการจัดแบ่งอุปกรณ์ออกเป็นคลาสต่างๆ ตามกำลังส่งสูงสุดของอุปกรณ์ ซึ่งจะเห็นได้ว่าระยะไกลสุดที่สามารถรับส่งข้อมูลได้มีระยะสูงถึง 100 เมตรสำหรับอุปกรณ์คลาส 1 แต่ในปัจจุบันอุปกรณ์ที่ผลิตออกมาส่วนใหญ่จะอยู่ในคลาส 3 เพราะมีกำลังส่งที่ไม่สูงมากซึ่งหมายถึงการประหยัดพลังงาน ทำให้สามารถนำอุปกรณ์เหล่านี้ไปใช้งานร่วมกับอุปกรณ์พกพาที่มีความต้องการอุปกรณ์ที่ใช้พลังงานน้อยเพื่อประหยัดพลังงานแบตเตอรี่

2.10 เทคนิคการส่งข้อมูล

เมื่อได้ข้อตกลงด้านความถี่ที่ใช้งาน, จำนวนช่องสัญญาณ และแบนวิดธ์ของแต่ละช่องสัญญาณมาเป็นกรอบแล้ว ต่อไปก็คือการเลือกวิธีมอดูเลทข้อมูลเข้าไปกับคลื่นพาหะ เนื่องจากการมอดูเลทแต่ละแบบจะส่งผลต่อความเร็วในการส่งข้อมูลด้วย และเนื่องจากความจำกัดในด้านแบนวิดธ์ของข้อมูลที่กว้างเพียง 1 MHz ต่อช่องสัญญาณ บวกกับความต้องการความเร็วในการส่งข้อมูลที่สูงที่สุด Bluetooth จึงได้เลือกใช้การมอดูเลทแบบ Gaussian Frequency – Shift Keying (GFSK)

การมอดูเลทด้วยวิธีนี้สามารถส่งข้อมูลได้ 1 บิตต่อความถี่คลื่นพาหะ 1 Hz นั่นหมายความว่าแต่ละช่องสัญญาณสามารถส่งข้อมูลได้ที่ความเร็ว 1 เมกะบิตต่อวินาที (Mbit/s) โดยถ้าบิตข้อมูลเป็น '1' จะเกิดการเปลี่ยนแปลงความถี่ทางด้านลบจากความถี่พาหะ

การรับ-ส่งข้อมูลจะทำโดยแบ่งข้อมูลออกเป็นแพ็กเกจย่อยๆ แล้วส่งในแบบฮาร์ฟดูเพล็กซ์ (Half Duplex) เพื่อประหยัดช่องสัญญาณ มิฉะนั้นจะต้องใช้ 2 ช่องสัญญาณเพื่อรับและส่งข้อมูลได้พร้อมๆ กัน จังหวะการรับ-ส่งข้อมูลทั้งหมดกำหนดโดยอุปกรณ์ที่เป็นมาสเตอร์ในลักษณะของการโพล ซึ่งอุปกรณ์ที่เป็นสเลฟจะต้องตอบกลับมายังมาสเตอร์ในทุกๆ แพ็กเกจ เพื่อให้มาสเตอร์รู้ว่ายังคงติดต่อกับสเลฟอยู่ได้

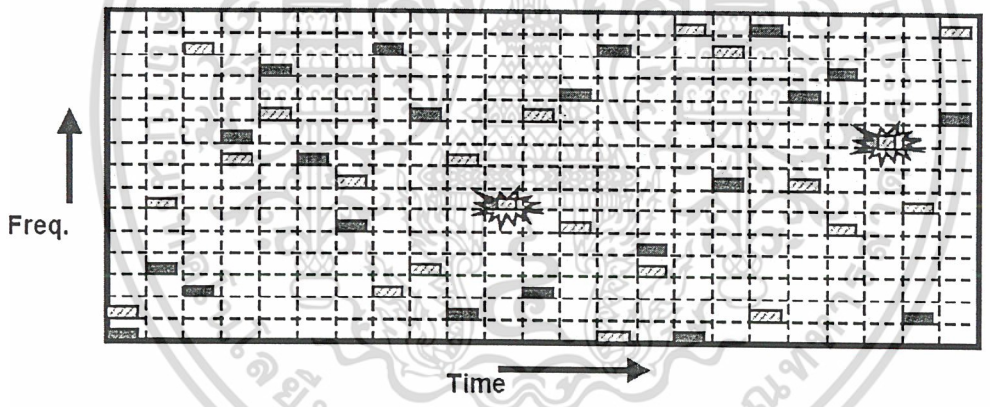
เมื่อมีการแบ่งข้อมูลออกเป็นแพ็กเกจ ทำให้แต่ละแพ็กเกจต้องมีข้อมูลส่วนหัว (Header) เพิ่มเข้ามา เพื่อให้ทางฝั่งรับสามารถประกอบข้อมูลทั้งหมดเข้าด้วยกันได้อย่างถูกต้อง นอกจากนั้นก่อนการส่งแต่ละครั้งจะต้องมีการส่งข้อมูลเพื่อทำการซิงโครไนซ์สัญญาณนาฬิกาทางฝั่งส่งและรับให้เท่ากัน เพื่อให้

สามารถรับและส่งข้อมูลกันได้อย่างถูกต้อง ซึ่งเมื่อรวมปริมาณข้อมูลทั้งหมดที่จำเป็นต้องส่งในแต่ละครั้ง จะทำให้ความเร็วในการส่งข้อมูลจริงลดลงจาก 1 Mbit/s เหลือ 723.2 Kbit/s ในทิศทางหนึ่งและ 57.6 Kbit/s ในอีกทิศทางหนึ่ง

นอกจากความเร็วและความสะดวกสบายแล้ว สิ่งที่จำเป็นอย่างยิ่งสำหรับการสื่อสารข้อมูลในปัจจุบันก็คือ ความปลอดภัยของข้อมูล โดยเฉพาะอุปกรณ์บลูทูธที่สามารถทำงานได้ในทุกที่ยิ่งมีความจำเป็นที่จะต้องมีการรักษาความปลอดภัยของข้อมูลเป็นอย่างดี เทคนิคการส่งข้อมูลที่บลูทูธใช้คือ เทคนิคการกระโดดข้ามทางความถี่ (Frequency Hopping Spread Spectrum : FHSS)

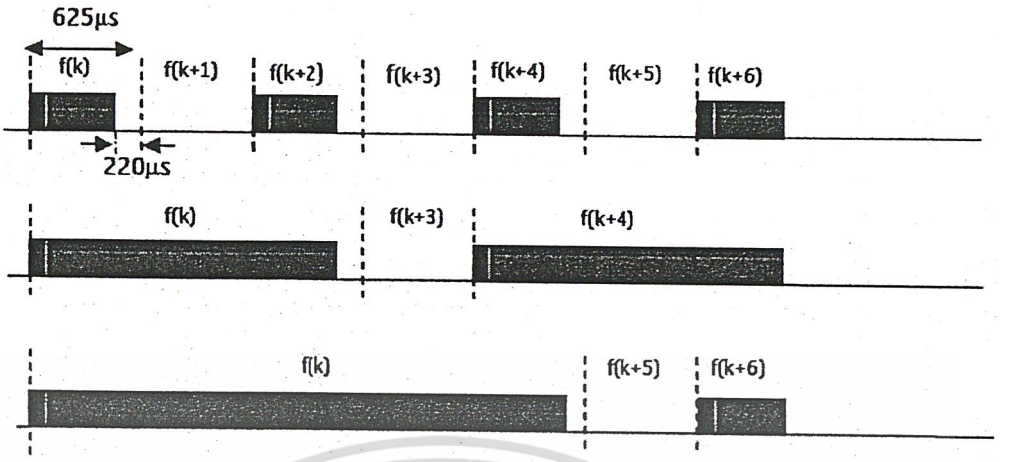
2.11 Frequency Hopping

เทคโนโลยีบลูทูธใช้เทคนิคของ Frequency Hopping ซึ่งทุก ๆ packet จะถูกส่งไปยังช่องความถี่ที่แตกต่างกัน ในบางประเทศอาจสามารถใช้งานได้มากถึง 79 ช่องสัญญาณ การต่อต้านสัญญาณรบกวนสามารถทำได้ดีเนื่องจากมี hop rate สูงถึง 1600 hops/sec เมื่อมีอุปกรณ์อื่นๆมารบกวนการส่ง packet นั้น ๆ จะถูกส่งใหม่ในอีกช่องสัญญาณ เช่นในรูปที่ 2.6 packet ของอุปกรณ์ 2 ตัวต้องการที่จะใช้ความถี่เดียวกัน



รูปที่ 2.6 Frequency Hopping และการเกิด collision

ในการรับและส่งสัญญาณจะมีการตัดแบ่งข้อมูลออกเป็นช่วงๆ ความยาวปกติคือ 625 ms โดยปกติแล้ว packet หนึ่งๆสามารถรวมอยู่ในช่อง (slot) เดียวได้ แต่ packet บางอันสามารถขยายได้ถึง 3 หรือ 5 slot ดังที่แสดงในรูปที่ 2.7 multi-slot packet ทุก ๆ slot จะถูกส่งในความถี่เดียวกันจนจบ packet นั้น และในการส่ง multi-slot packet จะมี data rate ที่สูงเพราะ header จะใช้งานเพียงครั้งเดียวใน 1 packet อย่างไรก็ดีในกรณีที่การส่งข้อมูลมีความหนาแน่น และ packet ที่มีความยาวมาก ก็จะมีโอกาสสูญเสียมาก



รูปที่ 2.7 Slot Packet

2.12 ภาครับ – ส่งสัญญาณวิทยุและสายอากาศ (Radio)

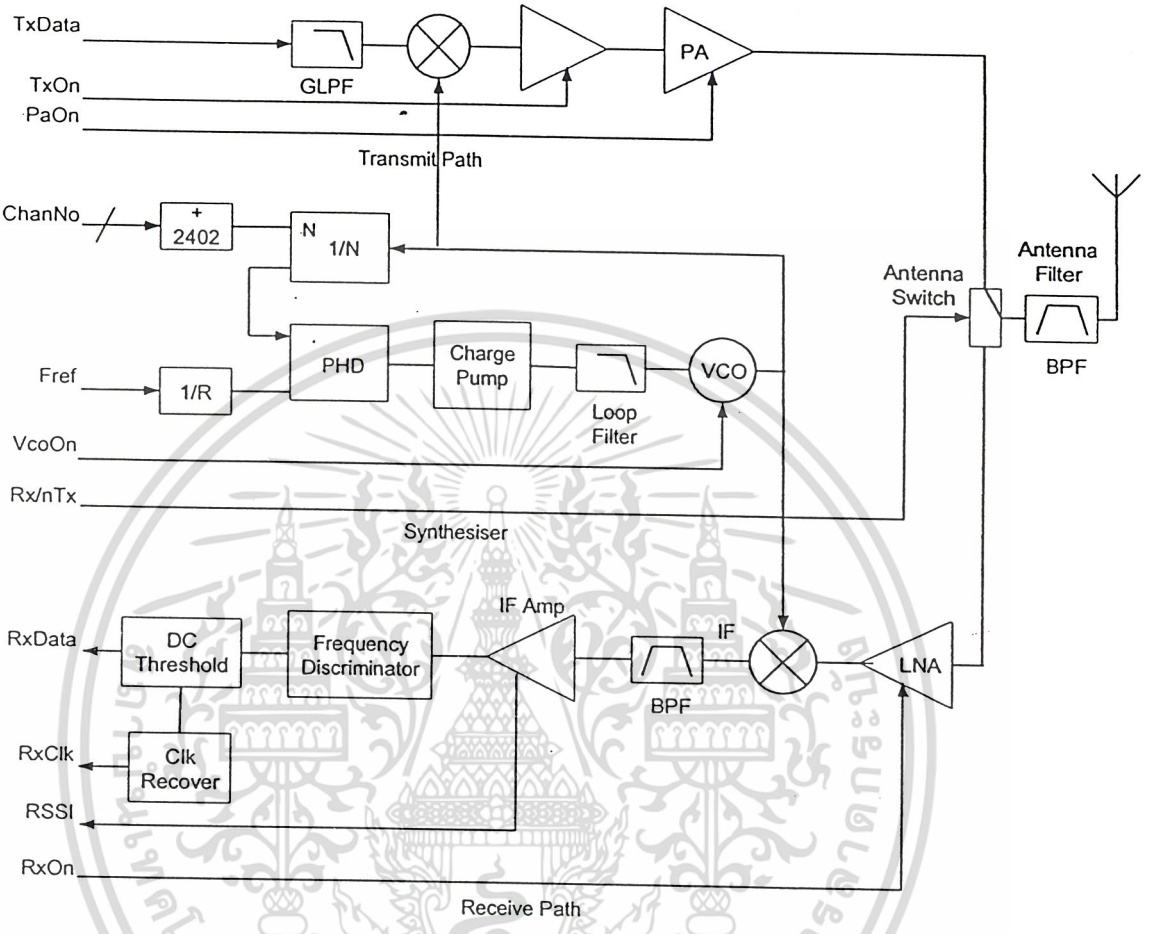
หน้าที่พื้นฐานของภาคนี้ก็คือ การรับและส่งสัญญาณวิทยุที่ผ่านการมอดูเลทข้อมูลเข้าไปแล้ว รูปที่ 2.8 แสดงให้เห็นบล็อกไดอะแกรมอย่างง่ายของโครงสร้างการทำงานภาครับ – ส่งสัญญาณวิทยุ

จากรูปที่ 2.8 จะเห็นได้ว่าสามารถแบ่งการทำงานทั้งหมดออกได้เป็น 3 ส่วน คือ ส่วนมอดูเลทข้อมูลภาคส่ง (บนสุด), ส่วนดีมอดูเลทข้อมูลภาครับ (ล่างสุด) และส่วนสร้างความถี่คลื่นพาหะ (กลาง) จะเห็นได้ว่าแต่ละส่วนจะมีการรับ – ส่งข้อมูลไปยังภายนอกซึ่งส่วนที่มาเชื่อมต่อนี้ก็คือภาคเบสแบนด์ (Baseband) นั่นเอง

ในส่วนของการมอดูเลทข้อมูล ในภาคส่งจะรับข้อมูลจากภาคเบสแบนด์มามอดูเลทกับความถี่พาหะด้วยวิธี GFSK (Gaussian Frequency – Shift Keying) หลังจากนั้นจะถูกนำไปขยายตามอัตราขยายที่ภาคเบสแบนด์กำหนด สุดท้ายก็จะถูกส่งออกอากาศทางสายอากาศ

ในส่วนของการดีมอดูเลท ในภาครับก็จะรับสัญญาณวิทยุจากสายอากาศแล้วส่งสัญญาณผ่านวงจรกรองสัญญาณแบบแบนด์พาส (Band Pass Filter) เพื่อเลือกเอาเฉพาะสัญญาณที่อยู่ในช่วงความถี่ ISM จากนั้นสัญญาณจะถูกป้อนเข้าสู่วงจรขยายสัญญาณ ซึ่งจะวัดระดับความแรงของสัญญาณส่งไปให้ภาคเบสแบนด์ด้วย และสุดท้ายจะนำข้อมูลที่ได้ไปสังเคราะห์เป็นสัญญาณนาฬิกาเพื่อนำไปใช้งานต่อไป

ในส่วนของการสร้างความถี่พาหะจะได้รับการเลือกช่องสัญญาณจากภาคเบสแบนด์ จากนั้นจะนำหมายเลขของช่องสัญญาณไปบวกด้วย 2,402 MHz ก็จะได้ค่าความถี่ที่ต้องการ จากนั้นก็จะนำค่าที่ได้นี้ไปป้อนให้แก่วงจรเฟสล็อกลูป (Phase Lock Loop) เพื่อสร้างความถี่ที่ต้องการออกมา



รูปที่ 2.8 โครงสร้างของภาครับ – ส่งสัญญาณวิทยุ

2.13 ภาคเบสแบนด์ (Baseband)

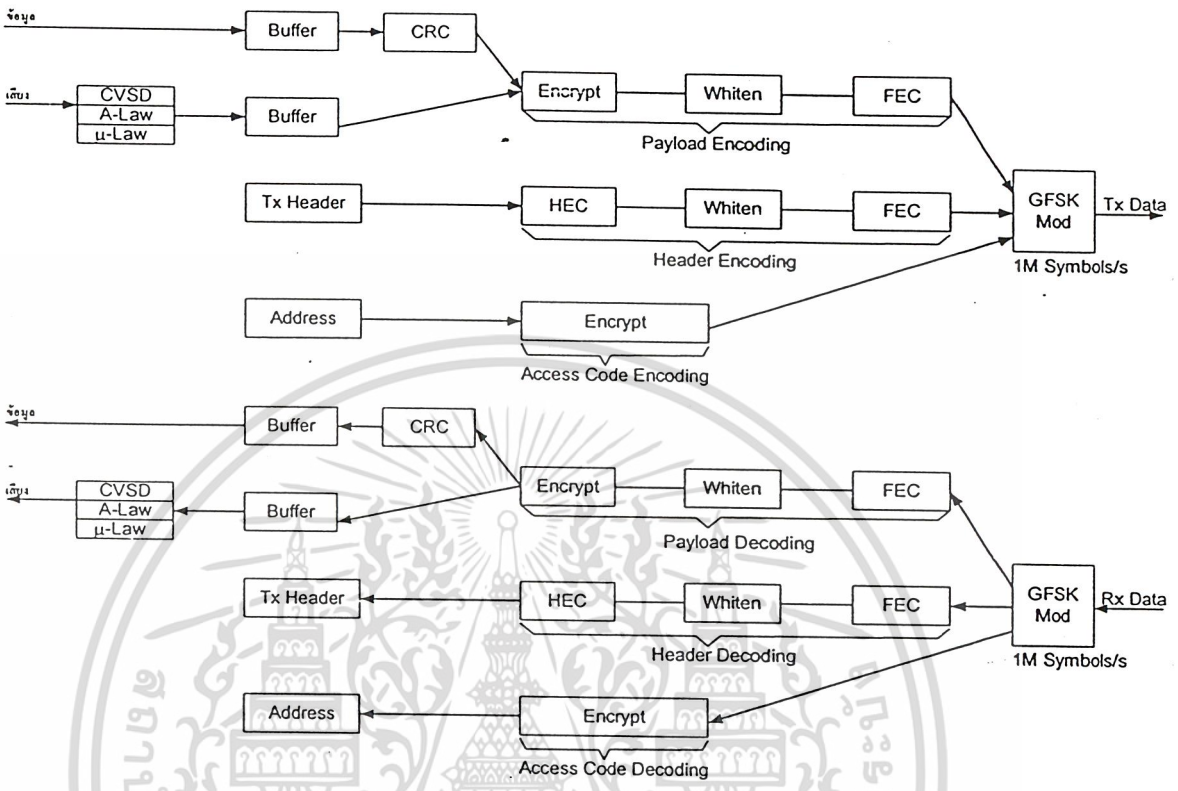
การทำงานทั้งหมดของภาครับ – ส่งสัญญาณคลื่นวิทยุจะถูกควบคุมโดยภาคเบสแบนด์ทั้งในด้านอัตราการขยายสัญญาณวิทยุและช่องความถี่ที่ใช้งาน นอกจากการควบคุมการทำงานของภาครับ - ส่งสัญญาณคลื่นวิทยุแล้ว อีกหน้าที่หนึ่งที่สำคัญก็คือ การสร้าง Access Code, Header และ ส่วนตรวจสอบความผิดพลาดรวมเข้ากับข้อมูลที่ต้องการส่งจริงเพื่อสร้างเป็นแพ็กเกจ

อัตราขยายของวงจรส่งจะถูกกำหนดให้มีค่าสูงสุด(ตามคลาสที่กำหนด)ในการส่งครั้งแรก แต่เมื่อมีการเชื่อมต่อกันสมบูรณ์แล้วอุปกรณ์ฝั่งรับจะส่งค่าความแรงของสัญญาณที่รับได้กลับไปให้อุปกรณ์ฝั่งส่ง ซึ่งอุปกรณ์ฝั่งส่งจะนำค่าที่ได้รับกลับมานี้ไปปรับค่าอัตราขยายไม่ให้สูงเกินจำเป็นเพื่อประหยัดพลังงาน ในส่วนของการเลือกช่องความถี่ในการส่งหรือรับนั้นจะคำนวณจากค่าแอดเดรสของอุปกรณ์ที่เป็นมาสเตอร์ในแต่ละพิโคเน็ต ซึ่งค่าแอดเดรสนี้จะถูกส่งออกมาจากมาสเตอร์ในส่วนของ Access Code

ในทุก ๆ แพ็กเกจข้อมูล นั่นหมายความว่าอุปกรณ์ทุกตัวที่อยู่ในพิโคเน็ต (เป็นมาสเตอร์ – สเลฟกัน) จะสามารถคำนวณเลือกช่องความถี่ที่เหมือนกันทำให้สามารถรับ – ส่งข้อมูลกันได้อย่างถูกต้อง ในขณะที่อุปกรณ์ในพิโคเน็ตอื่นก็จะมีรูปแบบการกระโดดที่แตกต่างออกไปเพราะค่าแอดเดรสของอุปกรณ์ที่เป็นมาสเตอร์ไม่เหมือนกัน (อุปกรณ์บลูทูธทุกตัวจะมีแอดเดรสของตัวเองที่ไม่ซ้ำกันเลย)

รูปที่ 2.9 แสดงให้เห็นถึงกระบวนการสร้างแพ็กเกจข้อมูลของภาคเบสแบนด์ ซึ่งจะเห็นได้ว่าแบ่งออกได้เป็น 3 ส่วนใหญ่ๆ คือ ส่วนของ Access Code, Header และ ส่วนข้อมูลจริง (Payload Data) โดยในขั้นตอนการสร้างแพ็กเกจข้อมูลจะมีบล็อกการทำงานย่อยๆ ได้แก่

- บล็อก CRC แสดงถึงการคำนวณส่วนป้องกันความผิดพลาดแบบ CRC-16
- บล็อก HEC แสดงถึงการคำนวณส่วนป้องกันความผิดพลาดแบบ CRC-8 ในส่วนของ Header
- บล็อก Whitening ข้อมูลที่หมายถึงการเติมบิตข้อมูลที่สุ่มอย่างมีรูปแบบเข้าไปในตัวข้อมูลเพื่อป้องกันการเกิดปัญหาจากการส่งบิตข้อมูล '1' หรือ '0' ต่อเนื่องกันเป็นจำนวนมากๆ
- บล็อก FEC (Forward Error Correction) เป็นการเข้ารหัสข้อมูลด้วยการเพิ่มพาริตีบิตแบบพิเศษเพื่อตรวจสอบความผิดพลาดในการส่ง และเมื่อพบว่าผิดพลาดก็สามารถแก้ไขข้อมูลให้กลับมาถูกต้องได้ (ถ้าไม่ผิดมากเกินไป) การเข้ารหัส FEC มีอยู่ 2 แบบ คือ แบบ 1/3 และ 2/3 โดยการเข้ารหัสแบบ 1/3 จะมีความแข็งแรงต่อความผิดพลาดมากกว่าแบบ 2/3 แต่ต้องเพิ่มพาริตีบิตเป็นจำนวนมากกว่า ใช้กับข้อมูลที่มีความสำคัญมาก เช่น Header เท่านั้น ส่วนข้อมูลทั่วไปถ้าจะเข้ารหัสจะใช้แบบ 2/3 เท่านั้น



รูปที่ 2.9 การสร้างแพ็กเกจข้อมูลในภาคเบสแบนด์

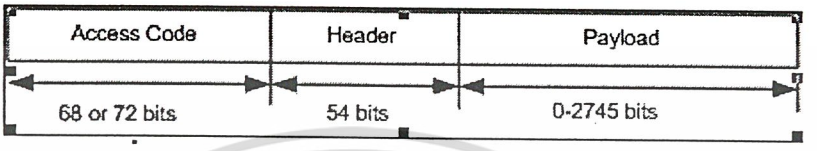
ส่วน Access Code เกิดจากการนำเอา Address ของอุปกรณ์มาผ่านการเข้ารหัสตามที่ระบุไว้ในมาตรฐานบลูทูธ มาสเตอร์หรือสเลฟที่ทำการรับข้อมูลบนเครือข่ายพิโกเน็ตจะทำการเปรียบเทียบสัญญาณที่เข้ามากับค่า Access Code ถ้า Access Code ไม่ตรงกันก็จะไม่พิจารณาแพ็กเกจที่รับเข้ามา ประโยชน์ของ Access Code นั้น นอกเหนือจากการใช้บ่งบอกเพื่อแยกแยะแพ็กเกจแล้ว ยังถูกใช้สำหรับการซิงโครไนซ์และการชดเชยสำหรับ offset อีกด้วย รหัส Access Code นี้ต้องมีคุณสมบัติที่ทนทานต่อสัญญาณแทรกแซงที่เข้ามารบกวนได้

ส่วนของ Header เป็นส่วนที่เก็บข้อมูลการควบคุมที่สำคัญ เช่น Media Access Control (MAC), Address, packet type, flow control bits, บิตที่ใช้สำหรับ ARQ (Automatic Retransmission Query) และ ฟิลด์ของ Header Error Check (HEC) Header ซึ่งมีความยาว 54 บิต เกิดจากการนำข้อมูลบ่งบอกลักษณะของแพ็กเกจและข้อมูลอื่นๆที่จำเป็นมาคำนวณส่วนป้องกันความผิดพลาดทั้งแบบCRC และ FEC

ส่วนของข้อมูล (payload) อาจจะมีหรือไม่มีก็ได้ จะสามารถแบ่งออกได้เป็น 2 ชนิด คือ ข้อมูลที่เป็นเสียงกับข้อมูลทั่วไป ซึ่งข้อมูลทั้ง 2 ชนิดจะมีกระบวนการสร้างแพ็กเกจแตกต่างกัน

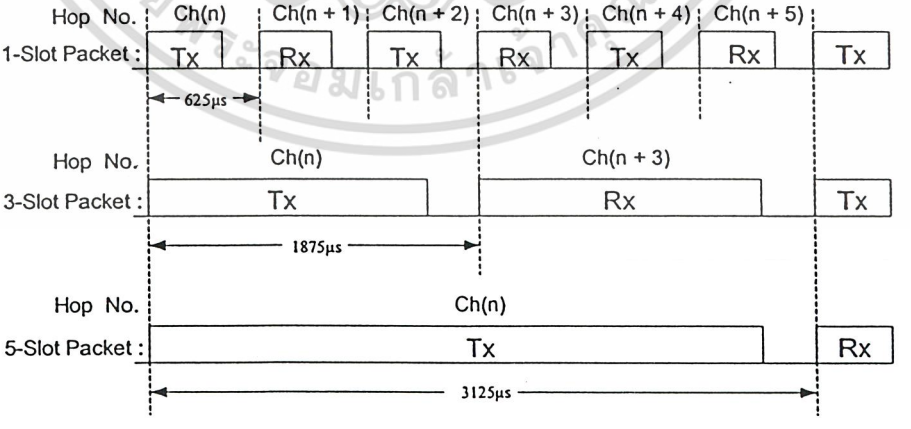
1 ขั้นตอน คือ ถ้าเป็นข้อมูลทั่วไป ตัวข้อมูลจะถูกนำไปคำนวณส่วนตรวจสอบความผิดพลาด CRC-16 ในขณะที่ข้อมูลเสียงจะไม่มี เนื่องจากการส่งข้อมูลเสียงจะเป็นการส่งแบบเรียลไทม์ ไม่สามารถส่งข้อมูลซ้ำได้ถึงแม้จะเกิดการผิดพลาดจึงไม่มีส่วนตรวจสอบนี้

ข้อมูลทั้ง 3 ส่วน เมื่อผ่านกระบวนการทั้งหมดแล้วก็จะถูกนำมารวมเพื่อสร้างเป็นแพ็กเกจ โดยส่วนแรกของแพ็กเกจคือ Access Code ตามมาด้วย Header และปิดท้ายด้วยตัวข้อมูลที่จะส่งจริงๆ รูปที่ 2.10 แสดงให้เห็นส่วนประกอบของข้อมูลใน 1 แพ็กเกจ



รูปที่ 2.10 ส่วนประกอบของแพ็กเกจข้อมูล

ถ้าหากต้องการส่งอัตราเร็วบิตข้อมูลที่สูง แพ็กเกจแบบมัลติสลอท (Multi Slot) จะถูกใช้ นั่นคือแพ็กเกจหนึ่งจะส่งด้วยการใช้ 1 สลอท, 3 สลอท หรือ 5 สลอท แต่แพ็กเกจหนึ่งต้องส่งด้วยการใช้ความถี่พาหะ (carrier) เดียวเสมอ ตัวอย่างเช่น ถ้า 4 สลอทที่เรียงต่อกันคือ k, k+1, k+2 และ k+3 และถูกกำหนดให้ใช้ hop frequencies คือ f_k, f_{k+1}, f_{k+2} และ f_{k+3} ถ้าเริ่มต้นส่งแพ็กเกจแบบ 3 สลอท สลอท k ก็จะส่งแพ็กเกจด้วยความถี่ f_k แพ็กเกจถัดไปก็จะเริ่มต้นใน slot k+3 และใช้ความถี่ f_{k+3} ดังรูปที่ 2.11 ซึ่งแสดงแพ็กเกจข้อมูลแบบมัลติสลอท



รูปที่ 2.11 multi slot แบบต่างๆ

เราสามารถแบ่งช่องสื่อสารการเชื่อมโยงได้เป็น 2 ประเภท คือ Synchronous Connection Oriented (SCO) link และ Asynchronous Connectionless (ACL) link

- การเชื่อมโยงแบบ SCO link เป็นการเชื่อมโยงจากจุดหนึ่งไปยังอีกจุดหนึ่ง(point to point) ที่อาศัยการสวิตช์วงจร (circuit switched) และมีการส่งข้อมูลที่สมมาตรกัน (symmetrical) ใน 2 ทิศทาง ปกติมักจะใช้สำหรับการสื่อสารของเสียง การเชื่อมโยงถูกกำหนดโดยการจองใช้ time slot ที่ติดกัน 2 slot (สำหรับ forward slot และ return slot) ทุกช่วงระยะเวลาห่างทางเวลาที่คงที่ โดยแพ็กเกจที่ใช้จะเป็นแบบ single slot และใช้ในการส่งเสียงด้วยความเร็ว 64 Kbit/s เสียงที่ส่งจะไม่ถูกป้องกันความผิดพลาด แต่ถ้าช่วงเวลาห่างในการส่งแพ็กเกจแต่ละครั้งถูกลดลง การแก้ไขความผิดพลาดโดย FEC แบบ 2/3 หรือ 1/3จะสามารถถูกเลือกนำมาใช้
- การเชื่อมโยงแบบ ACL link เป็นการเชื่อมโยงจากจุดหนึ่งไปยังหลายจุด (point to multipoint) ที่อาศัยการสวิตช์แพ็กเกจ (packet switched) และมีการส่งข้อมูลที่อาจจะสมมาตรกัน (symmetrical) หรือ ไม่สมมาตรกัน (asymmetrical) ใน 2 ทิศทางก็ได้ ปกติมักจะใช้สำหรับการส่งข้อมูลที่เป็นช่วงๆ (bursty data transmission) อุปกรณ์ที่เป็นมาสเตอร์จะใช้วิธีการโพล (polling) ในการเชื่อมโยงการควบคุมแบบ ACL ซึ่งแพ็กเกจข้อมูลอาจจะเป็นแบบ 1 slot, 3 slot หรือ 5 slot แพ็กเกจข้อมูลที่ส่งอาจจะไม่ถูกป้องกันความผิดพลาด หรือถูกป้องกันด้วย FEC แบบ 2/3 อัตราเร็วข้อมูลสูงสุด คือ 723.2 Kbit/s ในทิศทางหนึ่ง และ 57.6 Kbit/s ในทิศทางกลับกันนั้นสามารถได้รับจากการใช้แพ็กเกจแบบ 5 slot ที่ไม่ถูกป้องกันความผิดพลาด ตารางที่ 2.4 เป็นการสรุปอัตราเร็วบิตข้อมูลที่สามารถไดรับจาก ACL link โดย DMx แทนแพ็กเกจข้อมูลแบบ x slot ที่เข้ารหัส FEC ส่วน DHx จะแทนแพ็กเกจข้อมูลที่ไม่ถูกป้องกันความผิดพลาด

Type	Payload Header (bytes)	User Payload (bytes)	FEC	CRC	Symmetric Max. Rate (kb/s)	Asymmetric Max. Rate (kb/s)	
						Forward	Reverse
DM1	1	0-17	2/3	yes	108.8	108.8	108.8
DH1	1	0-27	no	yes	172.8	172.8	172.8
DM3	2	0-121	2/3	yes	258.1	387.2	54.4
DH3	2	0-183	no	yes	390.4	585.6	86.4
DM5	2	0-224	2/3	yes	286.7	477.8	36.3
DH5	2	0-339	no	yes	433.9	723.2	57.6
AUX1	1	0-29	no	no	185.6	185.6	185.6

ตารางที่ 2.4 อัตราเร็วบิตข้อมูลที่สามารถได้รับบน ACL link

2.14 HCI (Host Controller Interface)

ดังเช่นที่ได้อธิบายไปข้างต้นแล้วว่า HCI เป็นโปรโตคอลเชื่อมต่อระหว่างโปรแกรมชั้นบนที่ทำงานอยู่บนระบบหนึ่ง (เช่น โปรแกรมในเครื่องคอมพิวเตอร์โน๊ตบุ๊คทำงานบน CPU x86) กับส่วนควบคุมการทำงานของ Bluetooth (เช่น การ์ด PCMCIA Bluetooth ที่ต่ออยู่ในเครื่องคอมพิวเตอร์โน๊ตบุ๊ค) ทำให้โปรแกรมรู้จักคำสั่งควบคุมอุปกรณ์บลูทูธ

2.14.1 ชนิดของ HCI

บลูทูธได้แบ่งชนิดของ HCI ออกเป็น 3 แบบ ดังนี้

- HCI Command Packet ที่ถูกสร้างขึ้นโดยอุปกรณ์ควบคุม (Host) เพื่อนำไปควบคุมอุปกรณ์บลูทูธ (Host Controller) รูปแบบของ HCI Command Packet แสดงได้ในรูปที่ 2.12 อุปกรณ์บลูทูธแต่ละคำสั่งจะกำหนดค่า OpCode มา 2 ไบต์ แบ่งเป็น OpCode Group Field (OGF) และ OpCode Command Field (OCF) โดย OGF จะจองพื้นที่มากสุด 6 บิต ส่วน OCF จะจองพื้นที่ 10 บิต
- HCI Event Packet ถูกสร้างโดยอุปกรณ์บลูทูธเพื่อแจ้งให้โฮสท์ทราบถึงความเปลี่ยนแปลงที่เกิดขึ้นในอุปกรณ์บลูทูธ ซึ่งอุปกรณ์บลูทูธต้องสามารถรองรับ HCI Event Packet ได้ 255 ไบต์ โดยยกเว้นส่วนของ HCI Event Packet Header รูปแบบของ HCI Event Packet แสดงในรูปที่ 2.13
- HCI Data Packet ทำการสื่อสารข้อมูลหรือเสียงระหว่างอุปกรณ์บลูทูธและอุปกรณ์ควบคุม แบ่งเป็น HCI ACL Data Packet และ HCI SCO Data Packet

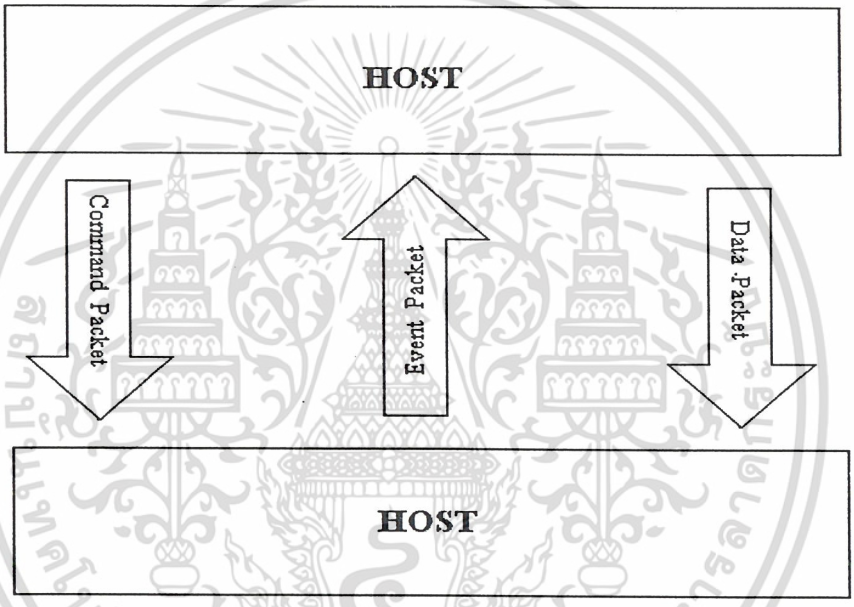
แผนภาพการทำงานของ HCI แต่ละแบบนั้น แสดงได้ดังรูปที่ 2.14

<table>
<tr><td colspan="2">OpCode</td><td rowspan="2">Parameter Total Length</td><td rowspan="2">Parameter 0</td></tr>
<tr><td>OCF</td><td>OCF</td></tr>
<tr><td colspan="2">Parameter 1</td><td colspan="2">Parameter 2</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Parameter N-1</td><td colspan="3">Parameter N</td></tr>
</table>

รูปที่ 2.12 HCI Command Packet

Event Code	Parameter Total Length	Event Parameter 0	
Event Parameter 1		Event Parameter 2	Event Parameter 3
Event Parameter N-1		Event Parameter N	

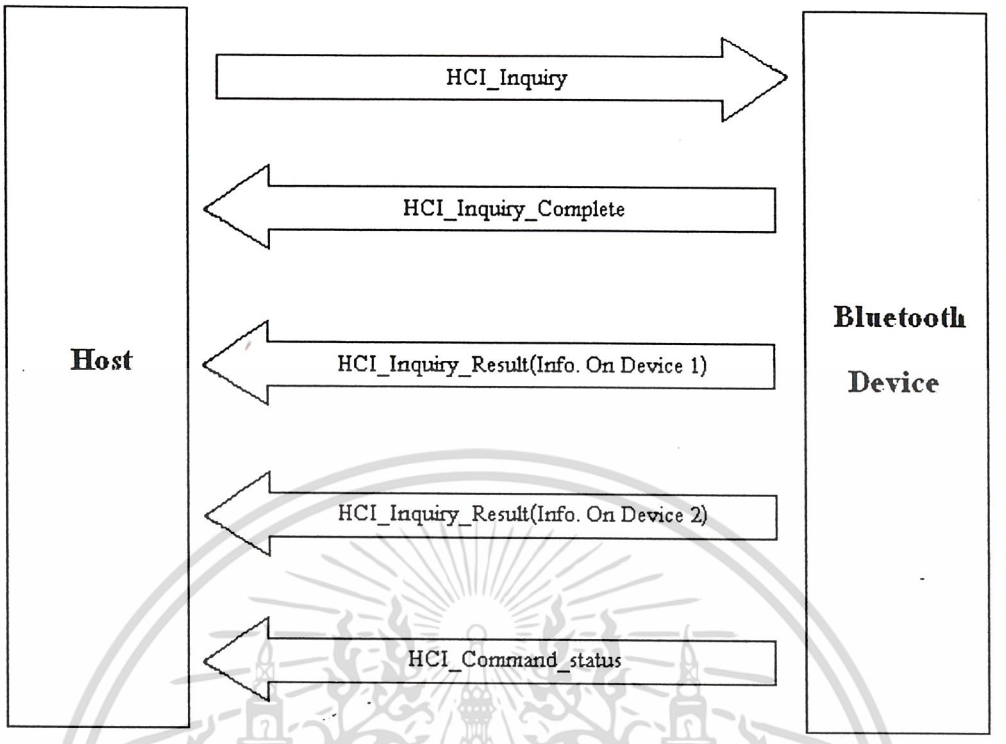
รูปที่ 2.13 HCI Event Packet



รูปที่ 2.14 HCI Command Packet

2.14.2 การค้นหาอุปกรณ์บลูทูธที่อยู่ใกล้เคียง

การค้นหาอุปกรณ์บลูทูธที่อยู่ใกล้เคียง จะตอบสนองการรับและการแสดงอุปกรณ์ โดยใช้คำสั่ง Host_Inquiry_Result ประกอบไปด้วยข้อมูลหลักคือ จำนวนของการตอบสนอง หรืออาจหมายถึงจำนวนของอุปกรณ์บลูทูธ การทำงานด้านการค้นหาอุปกรณ์เป็นดังรูปที่ 2.15

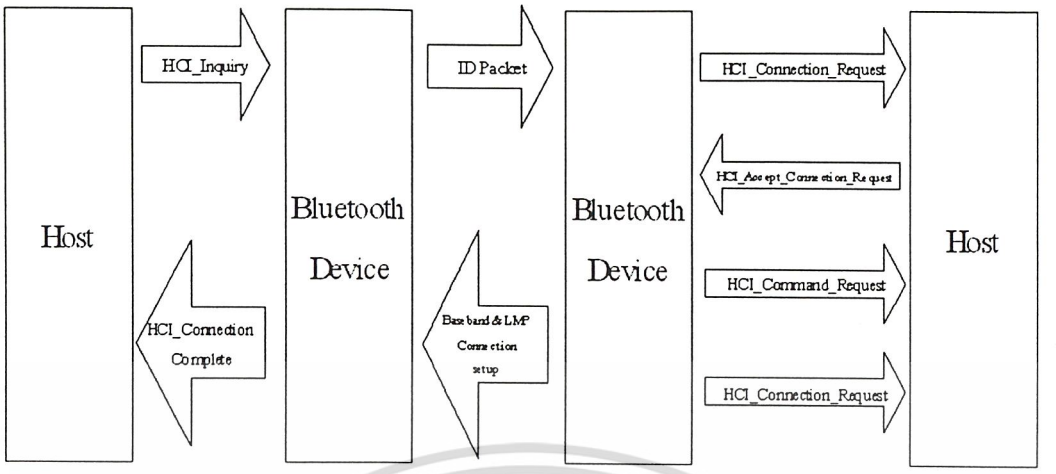


รูปที่ 2.15 การค้นหาอุปกรณ์บลูทูธ

โฮสท์จะทำการส่งข้อมูลการร้องขอการค้นหาอุปกรณ์ไปยังอุปกรณ์บลูทูธ(HCI_Inquiry) จากนั้นอุปกรณ์บลูทูธจะทำการค้นหาอุปกรณ์บลูทูธที่มีอยู่ใกล้เคียงและทำการแสดงผล (HCI_Inquiry_Complete) ของอุปกรณ์ต่าง ๆ ไปยังโฮสท์ เมื่อทำการค้นหาอุปกรณ์ครบทั้งหมดแล้วจะส่งข้อมูลแสดงการเสร็จสิ้นการค้นหาไปยังโฮสท์ (HCI_Inquiry_Complete)

2.14.3 การเชื่อมต่อกับอุปกรณ์บลูทูธ

การเชื่อมต่อกับอุปกรณ์บลูทูธทำได้โดยโฮสท์ทำการส่งคำสั่ง HCI_Create_Connection เพื่อทำการระบุข้อมูลที่ต้องใช้ในการเชื่อมต่อ โดยมีข้อมูลหลักดังนี้ คือ แอดเดรสของอุปกรณ์ที่จะทำการเชื่อมต่อ และหน้าที่โหมดการทำงานของอุปกรณ์ที่จะทำการเชื่อมต่อ (มาสเตอร์ หรือ สเลฟ) จากนั้นอุปกรณ์บลูทูธที่จะทำการถูกเชื่อมต่อจะได้รับ HCI_Connection_request เพื่อแจ้งให้ทราบถึงการร้องขอการเชื่อมต่อ และจะทำการตอบรับหรือปฏิเสธการเชื่อมต่อ ถ้าอุปกรณ์ บลูทูธที่ถูกร้องขอปฏิเสธการเชื่อมต่อจะส่งคำสั่ง HCI_Reject_Connection_request ไปยังอุปกรณ์บลูทูธที่ร้องขอการเชื่อมต่อ แต่ถ้าอุปกรณ์บลูทูธที่ถูกร้องขอตอบรับการเชื่อมต่อก็จะส่งคำสั่ง HCI_Accept_Connect_request ไปยังอุปกรณ์บลูทูธเพื่อทำการเซ็ตอัพการเชื่อมต่อ จากนั้นอุปกรณ์ บลูทูธทั้ง 2 ฝั่ง จะทำการส่งคำสั่ง HCI_Connection_Complete ไปยังโฮสท์ของตัวเอง เพื่อทำการแจ้งให้ทราบว่า การเชื่อมต่อสำเร็จแล้ว การทำงานจะเป็นไปดังรูปที่ 2.16

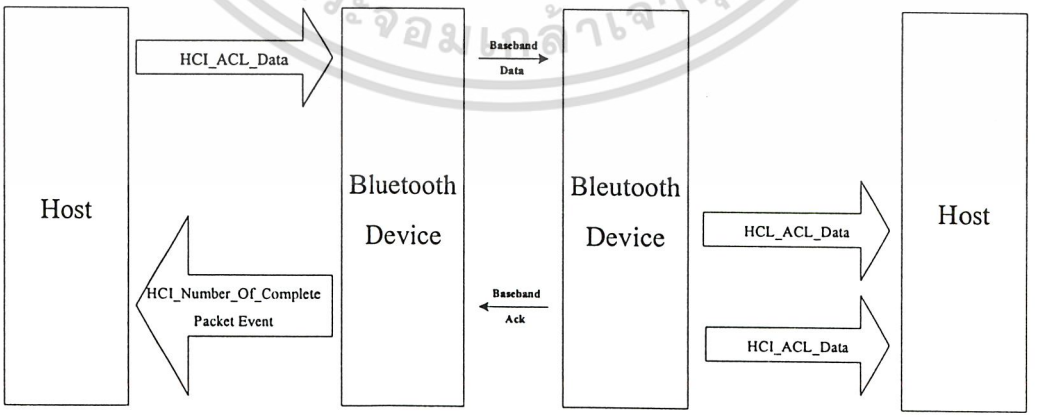


รูปที่ 2.16 การเชื่อมต่อระหว่างอุปกรณ์บลูทูธ

2.14.4 การส่งและการรับข้อมูล

การส่งข้อมูลจะแบ่งออกเป็น 2 ประเภท คือ การส่งข้อมูลแบบ ACL (Asynchronous Connectionless) ซึ่งเป็นการเชื่อมต่อสำหรับข้อมูลทั่วไป และการส่งข้อมูลแบบ SCO (Synchronous Connection-Oriented) เป็นการเชื่อมต่อที่ใช้สำหรับการส่งข้อมูลที่เป็นเสียง ในที่นี้จะขออธิบายการส่งและรับข้อมูลแบบ ACL เท่านั้น

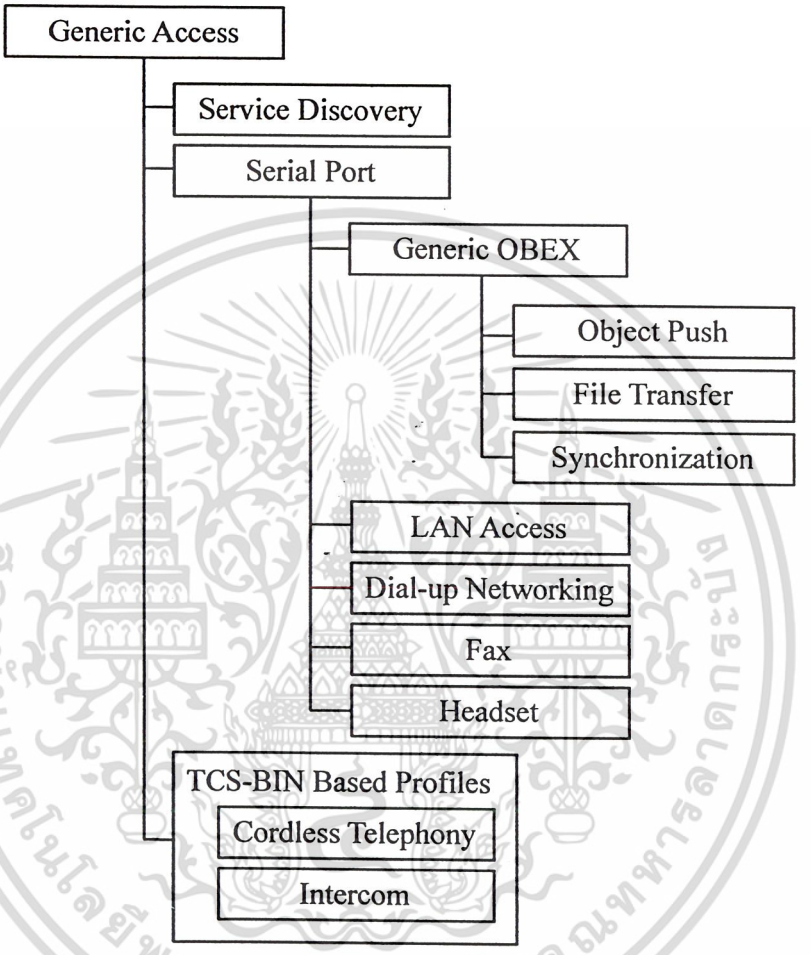
เมื่ออุปกรณ์บลูทูธทำการเชื่อมต่อสำเร็จแล้ว ถ้าอุปกรณ์ต้องการส่งข้อมูลโฮสท์จะทำการส่งHCI_ACL_Dataไปยังอุปกรณ์บลูทูธ จากนั้นอุปกรณ์บลูทูธอีกฝั่งหนึ่งจะทำการส่ง HCI_ACL_Data ไปยังโฮสท์ของตัวเอง จากนั้นอุปกรณ์บลูทูธที่เป็นตัวส่งสัญญาณจะทำการส่งจำนวนของข้อมูลการส่งที่สมบูรณ์ไปยังโฮสท์ของตัวเองในรูปของคำสั่ง HCI_Number_Of_Completed Packets Event โดยการทำงานเป็นไปดังรูปที่ 2.17



รูปที่ 2.17 การส่งและการรับข้อมูล

2.15 แอพพลิเคชั่นซอฟท์แวร์แอนด์บลูทูธโปรไฟล์ (Application Software and Bluetooth Profiles)

แอพพลิเคชันซอฟท์แวร์ (application software) เป็นส่วนประกอบในระดับสูง (high-level) ของบลูทูธ ประกอบด้วย user interface และ profile ต่าง ๆ มากมายที่ทำให้แอพพลิเคชันเข้าถึงการทำงานของฮาร์ดแวร์ (hardware) ได้ รูปที่ 2.18 แสดง profile ทั่วไปที่ใช้ในบลูทูธ



รูปที่ 2.18 บลูทูธโปรไฟล์ (Bluetooth Profile)

Bluetooth Profile ทำให้อุปกรณ์บลูทูธแต่ละชนิดติดต่อเชื่อมถึงกันได้ โดยมีมาตรฐานในการติดต่อและเข้าถึงตัวอุปกรณ์เช่นเดียวกัน ทำให้อุปกรณ์บลูทูธชนิดหนึ่งจากผู้ผลิตหนึ่งสามารถเชื่อมต่อกับอุปกรณ์บลูทูธอีกชนิดหนึ่งจากอีกผู้ผลิตหนึ่งได้ ตัวอย่างเช่น อุปกรณ์บลูทูธที่ติดตั้งมาภายในคอมพิวเตอร์แบบกระเป๋าหิ้วของบริษัท ก. สามารถเชื่อมต่อเข้ากับอุปกรณ์บลูทูธของที่ติดตั้งมาภายในโทรศัพท์เคลื่อนที่ของบริษัท ข. ได้ เป็นต้น

2.15.1 เจเนริคแอคเซสโปรไฟล์ (Generic Access Profile - GAP)

GAP เป็น profile ที่เป็นพื้นฐานของ profile อื่น ๆ ทั้งหมด กล่าวคือ profile อื่น ๆ จะต้องอาศัยหลักการเบื้องต้นของ GAP และเนื่องจาก profile นี้เป็นพื้นฐานของชุดอุปกรณ์ที่ใช้ติดต่อสื่อสารทั้งหมด ฉะนั้นแล้วในทุกๆอุปกรณ์จึงต้องสนับสนุนการทำงานของ profile นี้ ด้วยความต้องการพื้นฐานที่เหมือนกัน คือ

1. จะต้องสามารถใช้ได้กับอุปกรณ์บลูทูธทุก ๆ ชิ้น
2. ลักษณะการทำงานโดยทั่วไป คือ การค้นหาอุปกรณ์บลูทูธที่จะมาเชื่อมต่อด้วยให้พบ แล้วทำการเชื่อมต่อ
3. สามารถจัดการกับลิงค์ (link) ที่ใช้ในการเชื่อมต่อระหว่างอุปกรณ์บลูทูธได้อย่างสะดวกและง่าย
4. โพรซีเยอร์ (Procedure) ที่เกี่ยวข้องกับ profile นี้ จะต้องสามารถใช้ในระดับการรักษาความปลอดภัย (security level) ขั้นต่าง ๆ ได้ ซึ่งมีทั้งสิ้น 3 ระดับ ดังที่ได้กล่าวมาแล้ว

ด้วยเหตุนี้ วัตถุประสงค์ของ GAP คือ กำหนดหมวด (mode) การทำงานที่ใช้สำหรับ transport profile และ application profile และใช้เป็นตัวบอกสถานะการติดต่อ รวมถึงรับผิดชอบด้านลิงค์ (link) ที่มาเชื่อมต่อและช่องสัญญาณ (channel) ที่ใช้ติดต่อระหว่างอุปกรณ์บลูทูธอีกด้วย

2.15.2 เซอร์วิสดิสคัฟเวอร์รี่โปรไฟล์ (Service Discovery Profile – SDP)

โดยทั่วไปแล้วในอุตสาหกรรมอุปกรณ์บลูทูธจะคาดหวังถึงปริมาณและชนิดของบริการ (service) ที่สามารถรองรับการเชื่อมต่อระหว่างอุปกรณ์บลูทูธ ซึ่งเพิ่มขึ้นอย่างรวดเร็ว ฉะนั้นแล้วหน้าที่ของ SDP คือ จัดเตรียมโพรซีเยอร์ (Procedure) มาตรฐานสำหรับบริการชนิดต่าง ๆ ให้สามารถใช้งานร่วมกันกับอุปกรณ์บลูทูธทุกแบบได้

2.15.3 ซีเรียลพอร์ทโปรไฟล์ (Serial Port Profile)

profile นี้จัดเตรียมโปรโตคอล (protocol) และ โพรซีเยอร์ (procedure) สำหรับให้อุปกรณ์บลูทูธทำงานในลักษณะเดียวกันกับพอร์ทอนุกรม (serial port) ที่เชื่อมต่อด้วยสายอนุกรม (serial cable) ทั่วไป โดยจะทำการแปลงสัญญาณแบบ RS232 ก่อนที่จะทำการส่งสัญญาณไปยังปลายทาง เมื่อถึงปลายทางแล้วก็จะแปลงสัญญาณนั้นกลับไปเป็น RS232 เพื่อให้สามารถใช้งานได้เช่นเดียวกับพอร์ทอนุกรม (serial port) ทั่วไป

Serial Port Profile นี้ใช้ในการพัฒนา profile ย่อย ๆ อีก 5 profile ได้แก่

1. เฮดเซ็ทโปรไฟล์ (Headset Profile) เป็น profile ที่ใช้ในอุปกรณ์บลูทูธประเภทเฮดเซ็ท (headset) ที่ใช้กับโทรศัพท์เคลื่อนที่ ซึ่งไม่จำเป็นต้องสนับสนุนระบบรักษาความปลอดภัย (security) หรือการเข้ารหัส (encryption) ทั้งยังไม่จำเป็นต้องมีการกำหนดรูปแบบในลักษณะของมาสเตอร์ (master) และ สเลฟ (slave) เหมือนกับการติดต่อปกติ

2. แฟกซ์โปรไฟล์ (Fax Profile) เป็น profile ที่ให้บริการในการส่งข้อมูลในรูปแบบของ Fax ซึ่งไม่จำเป็นต้องมีการกำหนดรูปแบบในลักษณะของมาสเตอร์ (master) และ สเลฟ (slave) เช่นเดียวกับใน headset profile แต่สนับสนุนการเข้ารหัส (encryption) ข้อมูลที่จะทำการส่ง

3. ไดอัล – อัพเน็ทเวิร์คกิ้งโปรไฟล์ (Dial-up Networking Profile) เป็น profile ที่สนับสนุนการเชื่อมต่อกับอินเตอร์เน็ท (internet) โดยอาศัยอุปกรณ์บลูทูธเชื่อมต่อแทนโมเด็ม (modem) ซึ่งจำเป็นจะต้องใช้ระบบรักษาความปลอดภัย (security) อย่างสมบูรณ์แบบ เพื่อป้องกันข้อมูลที่ส่งผ่านระหว่างอุปกรณ์ขณะเชื่อมต่ออยู่

4. แลนแอคเซสโปรไฟล์ (LAN Access Profile) เป็น profile ที่สนับสนุนการเชื่อมต่อกันระหว่างอุปกรณ์บลูทูธในลักษณะเครือข่ายเน็ทเวิร์ครูปแบบแลน (LAN) ซึ่งทำให้สามารถเข้าถึงและใช้ข้อมูลร่วมกันระหว่างอุปกรณ์ได้

5. Generic Object Exchange Profile (GOEP) เป็น profile ที่ใช้เกี่ยวกับการแลกเปลี่ยน คือ ส่งหรือรับวัตถุ (Object) ต่าง ๆ ระหว่างอุปกรณ์บลูทูธ เช่น แฟ้มข้อมูล (file), เสียง (voice) เป็นต้น ซึ่งสามารถแบ่งย่อยออกไปได้อีก 3 profile ได้แก่ Object Push Profile, File Transfer Profile, Synchronization Profile

2.16 ส่วนควบคุมการเชื่อมต่อ

ส่วนควบคุมการเชื่อมต่อ Bluetooth device จะมีสถานะต่างๆ ดังนี้

1. Standby ไม่มีการทำงาน ไม่มีการส่งข้อมูลระหว่างอุปกรณ์ ถูกใช้ในช่วงที่อุปกรณ์มีการประหยัดพลังงาน

2. Inquiry ค้นหาอุปกรณ์ตัวอื่นและอุปกรณ์ที่จะติดต่อด้วย อุปกรณ์ที่ถูก Inquiry จะส่ง packet FHS จะมีข้อมูลสำคัญที่จะทำให้อุปกรณ์ที่ Inquiry นั้น สร้างการเชื่อมต่อขึ้นมา

3. Inquiry Scan คอยรับ packet inquiry ที่ใช้ในการเข้าถึง เมื่อได้รับสมบูรณ์แล้ว จะตอบสนองต่อ inquiry โดยใช้ข้อมูลจาก packet FHS ที่ได้รับมา

4. Page Master กับ Slave ส่งและตอบข้อความ page ถึงกัน

5. Page Scan อนุญาตให้ใช้อุปกรณ์ที่ทำการ paging สร้างการเชื่อมต่อ

6. การเชื่อมต่อ การทำงาน ข้อมูลถูกแลกเปลี่ยนซึ่งกันและกัน

7. การเชื่อมต่อ - พัก การรับส่งข้อมูลหยุดอยู่ในช่วงเวลาที่กำหนด เพื่อให้มี BW ว่างพอที่จะนำไปใช้ทำงานอื่น

8. การเชื่อมต่อ – sniff slave คอยฟังการสื่อสารที่เกิดขึ้น

9. การเชื่อมต่อ - หยุดรอ slave ยกเลิก AM_ADDRESS และคอยฟังการสื่อสารเป็นช่วงๆ

การทำงานของส่วนควบคุมการเชื่อมต่อ

1. Host ร้องขอการ inquiry

2. inquiry ถูกส่งไปโดยใช้ลำดับความถี่กระโดดที่ใช้ในการ inquiry

3. อุปกรณ์ที่ทำการ inquiry ตอบกลับด้วย packet FHS ประกอบด้วยข้อมูลในการสร้างการเชื่อมต่อ

4. ข้อมูลใน packet FHS ถูกส่งกลับไปยัง host

5. host ร้องขอการเชื่อมต่อไปยังอุปกรณ์ที่ตอบรับการ inquiry

6. กระบวนการ paging ใช้ในการเริ่มต้นการเชื่อมต่อไปยังอุปกรณ์ที่ต้องการติดต่อด้วย

7. อุปกรณ์ที่ต้องการติดต่อด้วยทำการ page scan อยู่จะตอบกลับมา

8. ถ้าอุปกรณ์ที่ทำการ page scan อยู่ ยอมรับการเชื่อมต่อจะกนระโดดไปใช้ความถี่ที่ master กำหนดให้และติดต่อสื่อสารผ่านลำดับความถี่ที่ใช้ในการกระโดดนั้น

ส่วนจัดการการเชื่อมต่อ

จัดการการทำงานต่างๆดังนี้

1. นำ slave ไปรวมกับ piconet และกำหนดเลข AM_ADDRESS ให้อุปกรณ์นั้น

2. หยุดการเชื่อมต่อชั่วคราว เพื่อนำ slave เพื่อรวมเข้าไปใน piconet

3. ปรับแต่งค่าการเชื่อมต่อ รวมถึงสลับหน้าที่ ระหว่าง master กับ slave

4. สร้างการเชื่อมต่อแบบ SCO และ ACL

5. นำการเชื่อมต่อไปยังโหมดประหยัดพลังงาน เช่น พัก, sniff, หยุดรอ

6. ควบคุมโหมดการทดลอง

ส่วนการจัดการเชื่อมต่อ Bluetooth นี้ จะติดต่อกับส่วนที่อยู่ใน Bluetooth device ตัวอื่นโดยใช้ Protocal จัดการการเชื่อมต่อ

ส่วนการจัดการการเชื่อมต่อมีหน้าที่เริ่มต้นการเชื่อมต่อ และคอยดูแลการเชื่อมต่อให้ Bluetooth device สามารถติดต่อสื่อสารกันได้ และเริ่มสร้างการเชื่อมต่อแบบ ALC จากนั้นข้อความ LMP จะถูกใช้ในการเชื่อมต่อ SCO บนการเชื่อมต่อ ACL เดิม ส่วนการจัดการเชื่อมต่อจะดูแลข้อมูลบนอุปกรณ์ที่เป็น slave ซึ่งได้กำหนด AM_ADDRESS ให้

2.17 โมเดลการใช้งาน

แนวความคิดที่จะนำบลูทูธ ไปใช้งานด้านต่างๆ มีดังนี้

1. คอมพิวเตอร์ไร้สาย

ในปัจจุบันคอมพิวเตอร์ 1 เครื่อง ต้องเชื่อมต่อกับอุปกรณ์ภายนอกเป็นจำนวนมาก ไม่ว่าจะเป็น พรินเตอร์ คีย์บอร์ด เมาส์ หรือ ลำโพง การเชื่อมต่อดังกล่าวในปัจจุบันนี้ก็ใช้สายสัญญาณเป็นตัวเชื่อมต่อเกือบทั้งหมด ซึ่งทำให้เกิดความลำบากทั้งในด้านการใช้งาน การเคลื่อนย้าย และ ความเป็นระเบียบ ถ้าเปลี่ยนการใช้สายไฟเชื่อมต่ออุปกรณ์ต่างๆ ทั้งหมดนี้มาเป็นบลูทูธ ก็จะทำให้ปัญหาดังกล่าวหมดไปทันที อีกทั้งยังง่ายต่อตัวผู้ใช้ เพราะการเชื่อมต่อทั้งหมด จะดำเนินไปโดยอัตโนมัติ ไม่ต้องกังวลเรื่องการต่อสายผิดตำแหน่งอีกต่อไป

2. ชุดหูฟัง

ในปัจจุบันการแก้ปัญหาของผู้ใช้โทรศัพท์ที่ต้องการใช้มือทั้งสองข้างทำงานอย่างอื่นไปพร้อมๆ กันด้วย คือการใช้ชุดหูฟัง โดยชุดหูฟังดังกล่าวจะมีสายเชื่อมต่อจากตัวโทรศัพท์มายังหูฟังที่ติดอยู่กับตัวผู้ใช้ นั่นหมายความว่าตัวผู้ใช้ไม่สามารถเคลื่อนตัวไปไหนได้ไกลกว่าที่สายจะยาวถึงแล้วก็ต้องคอยระวังสายไม่ให้ไปเกี่ยวกับสิ่งต่างๆ ซึ่งจะทำชุดหูฟังหลุดออกจากตัวผู้ใช้ แต่เมื่อทดแทนสายที่กล่าวถึงนี้ด้วยบลูทูธ ผู้ใช้สามารถขยับตัวไปไหนได้อย่างสะดวก โดยไม่ต้องคอยระวังสายอีกต่อไป ถ้าในกรณี

สมอลทอร์กของโทรศัพท์มือถือ ผู้ใช้สามารถเก็บโทรศัพท์ไว้ที่ไหนก็ได้ใกล้ ๆ ตัวในขณะที่ใช้หูฟังบลูทูธ และ เนื่องจากชุดหูฟังบลูทูธไม่ได้เชื่อมต่อตายตัวกับอุปกรณ์ใด ดังนั้น จึงเป็นไปได้ว่าหูฟังตัวเดียวกันนี้ เมื่อไม่ได้ใช้งานเป็นสมอลทอร์ก ก็สามารถฟังเพลงจากเครื่องเล่น ซีดี หรือเป็นชุดหูฟังของโทรศัพท์บ้านอีกเครื่องก็ได้

3. อินเทอร์เน็ต บริดจ์

ในปัจจุบันหากต้องการเชื่อมต่ออุปกรณ์พกพาต่างๆ ไม่ว่าจะเป็นคอมพิวเตอร์โน้ตบุ๊ค หรือพ็อกเก็ตพีซีเข้ากับอินเทอร์เน็ต จำเป็นจะต้องเชื่อมต่ออุปกรณ์ดังกล่าวเข้ากับช่องทางสื่อสาร ซึ่งอาจจะเป็นโทรศัพท์มือถือหรือสายโทรศัพท์ธรรมดาผ่านทางสายเชื่อมต่อ แต่การเชื่อมต่อดังกล่าว สามารถใช้งานได้เพียง 1 อุปกรณ์ ต่อ 1 ครั้ง และยังเกิดปัญหาในเรื่องความเกะกะของสายเมื่อต้องใช้งานนอกสถานที่หรือในยานพาหนะต่างๆ แต่เมื่อทดแทนด้วยบลูทูธ เครื่องคอมพิวเตอร์ หรือ อุปกรณ์พกพาต่างๆ จะสามารถเชื่อมต่อเข้ายังอินเตอร์เน็ตได้ โดยต่อผ่านโทรศัพท์มือถือที่มีระบบ GPRS โดยไม่จำเป็นต้องใช้สาย ซึ่งจะช่วยลดความยุ่งยาก อีกทั้งยังเพิ่มความสะดวกสบายในการทำงานขึ้นด้วย

2.18 JSR – 82 The Java Bluetooth API

JSR – 82 เป็น Office Java Bluetooth API ซึ่งถูกกำหนดมาตรฐานโดย JSR – 82 Expert Group JSR – 82 จะประกอบไปด้วย packet จำนวน 12 packet คือ

1. javax.bluetooth ประกอบไปด้วย 13 class สำหรับสร้างการสื่อสาร Bluetooth protocol
2. javax.obex ประกอบไปด้วย 8 class สำหรับใช้ในการส่งผ่าน Object ระหว่างอุปกรณ์

OBEX protocol ถูกใช้สำหรับส่ง object ไปมาระหว่างอุปกรณ์มาเป็นเวลานานแล้วด้วย Infrared Technology ซึ่ง Bluetooth ได้ทำการดัดแปลง protocol นี้มาใช้ในการส่ง object ด้วยเช่นกัน และเนื่องจาก JSR-82 เป็น Bluetooth API อย่างเป็นทางการของภาษาจาวา ดังนั้นผู้ผลิตทุกๆ รายที่จะนำเอาไปใช้ จะต้องทำการรวม layer และ profile มาตรฐานตามที่กำหนด รวมไว้ใน SDK ของตนด้วย

Bluetooth SDK ที่ใช้ JSR-82 จะต้องมี Bluetooth Stack Layer มาตรฐานดังต่อไปนี้

1. Host Controller Interface (HCI)
2. Logical Link Control and Adaption Protocal (L2CAP)
3. Service Discovery Protocal (SDP)
4. RFCOMM

และจะต้องมี profile ดังต่อไปนี้

1. Generic Access Profile
2. Service Discovery Application Profile
3. Serial Port Profile
4. Generic Object Exchange Profile

ส่วนประกอบพื้นฐานสำหรับ Bluetooth Application จะต้องประกอบด้วย

1. Stack Initialization
2. Device Management
3. Device Discovery
4. Service Discovery
5. Service Registration
6. Communication

ใน Java Bluetooth Specification ได้มีการเพิ่มส่วนพิเศษเรียกว่า Bluetooth Control Center (BCC) เข้าไป ซึ่งในผู้ผลิตบางรายได้ใช้ BCC ในการทำ Stack Initialization

Stack Initialization

เป็นการเตรียม Bluetooth Device ให้พร้อมสำหรับการสื่อสารไร้สาย ลำดับในการตั้งค่าเริ่มต้นให้กับ stack จะไม่แน่นอน อาจขึ้นอยู่กับ OS ในบางกรณีอาจไม่ต้องเขียน code ในการตั้งค่าเริ่มต้นเลย หรือ ในอีกกรณีอาจจะต้องเขียน code เล็กน้อยในการ Baud rate สำหรับ interface แบบ RS – 232

Device Management

เป็น class ใน Java Bluetooth Specification ซึ่งมาจาก Generic Access Profile ซึ่งอนุญาตให้สามารถทำการบริหาร Bluetooth Device ประกอบด้วย class จำนวน 3 class คือ

a. javax.bluetooth.LocalDevice เป็น class ที่ให้ข้อมูลของตัวมันเอง (Local Bluetooth Device)

- getBluetoothAddress ใช้สำหรับเอา Bluetooth address ซึ่งมีลักษณะเหมือนกับ MAC address ออกมา ซึ่งจะเป็นอักขระจำนวน 12 หลัก
- setDiscoverable(int mode) ใช้สำหรับตั้งให้อุปกรณ์อื่นๆมองเห็นตัวมัน โดยจะมีอยู่ด้วยกัน 3 mode คือ
  - NOT_DISCOVERABLE ไม่มีอุปกรณ์ใดมองเห็น
  - GIAC อนุญาตให้ทุกๆอุปกรณ์มองเห็น
  - LIAC จะมองเห็นแบบชั่วคราวเป็นเวลา 1 นาที แล้วกลับไปเป็น state ก่อนหน้า
- getDiscoverable ใช้สำหรับเรียก discovery mode ปัจจุบันออกมา

b. javax.bluetooth.RemoteDevice ใช้สำหรับให้ข้อมูลเกี่ยวกับ Bluetooth device เครื่องอื่นๆ ในรัศมีการทำงาน

- getBluetoothAddress ใช้เรียก address ของ RemoteDevice
- getFriendlyName (Boolean alwayAsk)ใช้เรียกชื่อของ RemoteDevice เช่น " Andrew's PDA " เป็นต้น

c. javax.bluetooth.DeviceClass ใช้ในการแยกแยะประเภทของอุปกรณ์ในรัศมีทำการ ซึ่งจะแบ่งเป็น 2 ระดับ คือ Major Class และ Minor Class

- getMajorDeviceClass ใช้เรียก Major Class ของ RemoteDevice
- getMinor DeviceClass ใช้เรียก Minor Class ของ RemoteDevice

Device Discovery

ใช้สำหรับค้นหาว่ามีอุปกรณ์ชนิดใดที่อยู่ในรัศมีทำการบ้างในการใช้งาน Device discovery ถ้าเป็นการติดต่อแบบ peer – to – peer ฝ่ายใดฝ่ายหนึ่งจะเป็นผู้ใช้งาน แต่ถ้าเป็นแบบ client – server ฝั่ง client จะเป็นผู้เรียกใช้งาน

a. javax.bluetooth.DiscoveryAgent

- startInquiry (int accessCode, DiscoveryListener listener) ใช้สั่งให้ค้นหาอุปกรณ์อื่นๆในรัศมีทำการ
- retrieveDevice(int option) ใช้ในการเรียกดูรายการของอุปกรณ์ที่ได้จากจากคำสั่ง inquiry

b. javax.bluetooth. DiscoveryListener ถูกเรียกใช้โดย JVM เมื่อมีเหตุการณ์เกิดขึ้น

- deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) ถูกเรียกโดย JVM เมื่อพบ Bluetooth device จากการ inquiry

Service Discovery

ในการค้นหา service ของ remote device โดยการใช้งาน Service Discovery Protocal (SDP) layer ใน Bluetooth stack ในการค้นหา service ในแต่ละอุปกรณ์

a. javax.bluetooth.UUID UUID (Universal Unique Identifier) เป็น class ที่ทำการระบุ service ใน Bluetooth protocol เช่น L2CAP จะมี UUID เป็น 0x0001 เป็นต้น

- searchservice(int[ ] attrSet, UUID[ ]uuidSet,RemoteDevicebtDev,DiscoveryListen ใช้ในการค้นหารายการของ service ของ remote device ตัวเดียว
- selectService (UUID uuid, int security, Boolean master) ใช้ในการค้นหารายการของ service ของ remote device ตัวใดก็ได้ในรัศมีทำการ

b. javax.bluetooth.DiscoveryListener

- serviceDiscovered (int transID, ServiceRecode[ ] servRecode[ ]) ถูกเรียกโดย JVM เมื่อพบ service ใน remote device

c. javax.bluetooth.ServiceRecode

object ของ class นี้แสดงค่าแต่ละสมาชิกใน Service Discovery Database (SDDB) ซึ่งรวม ServiceRecode เอาไว้

d. javax.bluetooth.DataElement

service recode ที่เก็บอยู่ใน SDDB จะมี attribute เป็น object ของ class นี้

Service Registration

ก่อนที่ client จะสามารถใช้งาน service discovery ใน Bluetooth service device ได้ ทางฝั่ง server จะต้อง register service เสียก่อน ซึ่งจะเรียกว่า Service Registration

สิ่งที่เกิดขึ้นในการ Service Registration และบันทึกลงใน SDDB

1. เรียก connection.open ( ) ซึ่งทำให้เกิดการเชื่อมต่อกับ StreamConnectionNotifier object ซึ่ง connection.open ( ) จะสร้าง Servicerecode และ attribute ใหม่
2. ใช้ LocalDevice object และ StreamConnectionNotifier ในการเอา ServiceRecode ที่ถูกบันทึกโดยระบบมา
3. เพิ่มเติมหรือปรับปรุง attribute ใน ServiceRecode
4. ใช้ StreamConnectionNotifier ในการเรียก acceptAndOpen( ) และรอให้ client พบ Service และทำการเชื่อมต่อ
5. ระบบสร้าง ServiceRecode ใน SDDB รอจนกระทั่ง client เชื่อมต่อเมื่อ server พร้อมที่จะออกจากการทำงานจะเรียกใช้ close ใน StreamConnectionNotifier
6. ระบบลบ ServiceRecode ออกจาก SDDB

Communication

RFCOMM connection

รู้จักโดยทั่วไปว่าเป็น wireless serial port ซึ่งเป็น protocol ที่ใช้ในการใช้สาย cable ซึ่ง RFCOMM จะจำลอง function ต่างๆของ serial port มาตรฐาน เนื่องจาก Bluetooth Profile ส่วนมากใช้ serial port profile เป็น layer พื้นฐาน ทำให้โปรแกรมที่เดิมเคย ถูกออกแบบให้ใช้ serial port ในการเชื่อมต่อ สามารถใช้ bluetooth เชื่อมต่อแทนได้ ตัวอย่างเช่น การที่ PDA ทำการ Synchronize ข้อมูลให้กับ PC โดยใช้ RFCOMM ซึ่งเดิมเคยทำงานผ่านสาย cable

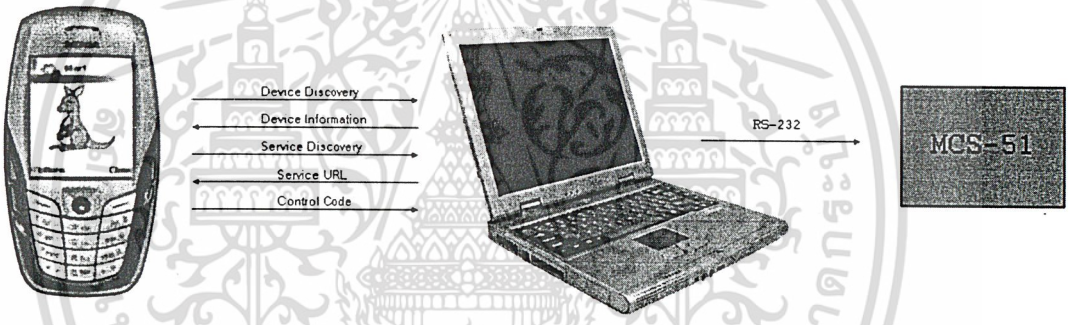
จะมีลักษณะเป็น packet oriented ซึ่งจะแตกต่างจาก RFCOMM ที่เป็น stream oriented ในการส่งข้อมูล Bluetooth API กำหนดให้สามารถกำหนดขนาดของ packet ได้เอง โดยจะเรียกขนาดที่มากที่สุดว่า Maximum Transmission Unit (MTU) ซึ่งค่า detail ของ MTU คือ 672 byte แต่ขนาดของ MTU ก็ยังสามารถขยายได้โดยการเจรจากันระหว่างฝ่ายรับและฝ่ายส่ง javax.bluetooth.L2CAPConnection เป็น subclass ของ Connection interface ซึ่งจะมี mothod เพิ่มเติมจากใน Connection ดังนี้

1. getReceiveMTU ใช้ในการเรียกค่า ReceiveMTU ที่เจรจาแล้วจาก Connection
2. getTransmitMTU ใช้ในการเรียกค่า TransmitMTU ที่เจรจาแล้วจาก Connection
3. Ready ( ) จะเป็นค่าที่เป็นจริงเมื่อข้อมูลพร้อมที่จะได้รับการอ่าน
4. receive (byte [ ] inBuf) ใช้กำหนดขนาดของ Buffer ในการรับข้อมูล ซึ่งจะต้องมีขนาดไม่น้อยกว่า Receive MTU
5. send (byte [ ] data) ใช้ส่งข้อมูลไปที่ remote Bluetooth device โดยที่ใช้ L2CAP protocal

บทที่ 3

การออกแบบและการสร้าง

โครงงานนี้ได้แบ่งการพัฒนาออกเป็น 3 ส่วน คือ ส่วนโปรแกรมควบคุมการเปิด-ปิดเครื่องใช้ไฟฟ้าบนมือถือ , ส่วนโปรแกรมบนเครื่องคอมพิวเตอร์และส่วนวงจรตัดต่อกระแสไฟฟ้าในห้องจำลอง ในการพัฒนาส่วนโปรแกรมควบคุมการเปิด-ปิดเครื่องใช้ไฟฟ้าบนมือถือและส่วนโปรแกรมบนเครื่องคอมพิวเตอร์ได้ใช้ภาษา JAVA และ JSR-82 Java Bluetooth API เป็นเครื่องมือในการพัฒนาโปรแกรม และส่วนสุดท้ายคือส่วนสวิตซ์ตัดต่อกระแสไฟฟ้าในห้องจำลอง ส่วนนี้ได้ทำการเชื่อมต่อกับเครื่องคอมพิวเตอร์โดยผ่านไมโครคอนโทรลเลอร์ MCS-51 และยังเชื่อมต่อกับห้องจำลองอุปกรณ์ไฟฟ้าเพื่อทดสอบการทำงานของโปรแกรม ระบบจำลองการควบคุมการเปิด-ปิดเครื่องใช้ไฟฟ้าภายในบ้านผ่านบลูทูธแสดงได้ดังรูป 3.1



รูปที่ 3.1 การทำงานโดยรวมของระบบจำลองการควบคุมการเปิด-ปิดเครื่องใช้ไฟฟ้าผ่านบลูทูธ

3.1 ส่วนโปรแกรมควบคุมเครื่องใช้ไฟฟ้าบนมือถือ

โปรแกรมนี้สามารถแบ่งออกได้เป็น 3 ส่วนหลัก คือ ส่วน Interface ที่ติดต่อกับผู้ใช้, ส่วนการเชื่อมต่อด้วยบลูทูธและส่วนการเลือกรหัสคำสั่ง

3.1.1 ส่วน Interface

โปรแกรมส่วนนี้เป็นส่วนที่ใช้ติดต่อกับผู้ใช้มีลักษณะเป็น GUI (Graphical User Interface) ซึ่งใช้ในการแสดงผลการดำเนินการต่างๆและรายการต่างๆที่ต้องการให้ผู้ใช้เลือก ซึ่งรายการส่วนใหญ่จะมีลักษณะเป็น choice group ซึ่งทำให้ผู้ใช้สามารถที่จะเลือกรายการต่างๆบนหน้าจอได้จากปุ่มทิศทางบนโทรศัพท์มือถือ รายการที่จะแสดงในส่วนนี้จะมีอยู่ 2 โหมดรายการคือ ชื่อของอุปกรณ์บลูทูธที่ค้นเจอและรายการอุปกรณ์ไฟฟ้าที่จะทำการเปิด-ปิด

3.1.2 ส่วนการเชื่อมต่อด้วยบลูทูธ

ส่วนนี้จะแบ่งการทำงานออกเป็น 3 ส่วน คือ

1. ส่วนการค้นหาอุปกรณ์ (Device Discovery)

ส่วนนี้จะเริ่มทำงานโดยอัตโนมัติเมื่อโปรแกรมถูกเรียกขึ้นมา ระบบจะทำการตรวจสอบสถานะของอุปกรณ์บลูทูธในเครื่อง จากนั้นจะทำการค้นหาอุปกรณ์บลูทูธอื่นๆที่อยู่ในรัศมีทำการ (10 เมตร) และนำชื่ออุปกรณ์บลูทูธที่ค้นพบส่งไปยัง user interface เพื่อให้ผู้ใช้ทำการเลือกอุปกรณ์บลูทูธที่จะทำการเชื่อมต่อด้วย

2. ส่วนการค้นหาบริการ (Service Discovery)

ส่วนนี้จะทำงานโดยอัตโนมัติเช่นกันแต่จะทำงานต่อจากการค้นหาอุปกรณ์ซึ่งทำการค้นหาโดยการเรียกดูบริการที่มี UUID (Universal Unique Identifier) ตรงกับที่กำหนดในอุปกรณ์ที่ค้นพบจากส่วนการค้นหาอุปกรณ์

UUID คือ เลขรหัสซึ่งระบุถึงบริการในอุปกรณ์บลูทูธใดๆ บริการทุกบริการจะมี UUID เป็นของตนเอง ซึ่งในบริการมาตรฐานนั้น UUID สามารถดูได้จาก https://www.Bluetooth.org/foundry/assignnumb/docment/assigned_numbers แต่เนื่องจากโปรแกรมการควบคุมเครื่องใช้ไฟฟ้ายังไม่มี การกำหนด UUID มาตรฐานขึ้นมาจึงใช้ UUID ซึ่งตั้งขึ้นมาเองซึ่งเป็น 1122334455 เมื่อโปรแกรมเข้าทำการค้นหาบริการที่มี UUID ดังกล่าวในอุปกรณ์บลูทูธ ถ้าค้นพบแล้วจะได้ค่า URL ซึ่งใช้ในการเชื่อมต่อกลับมา

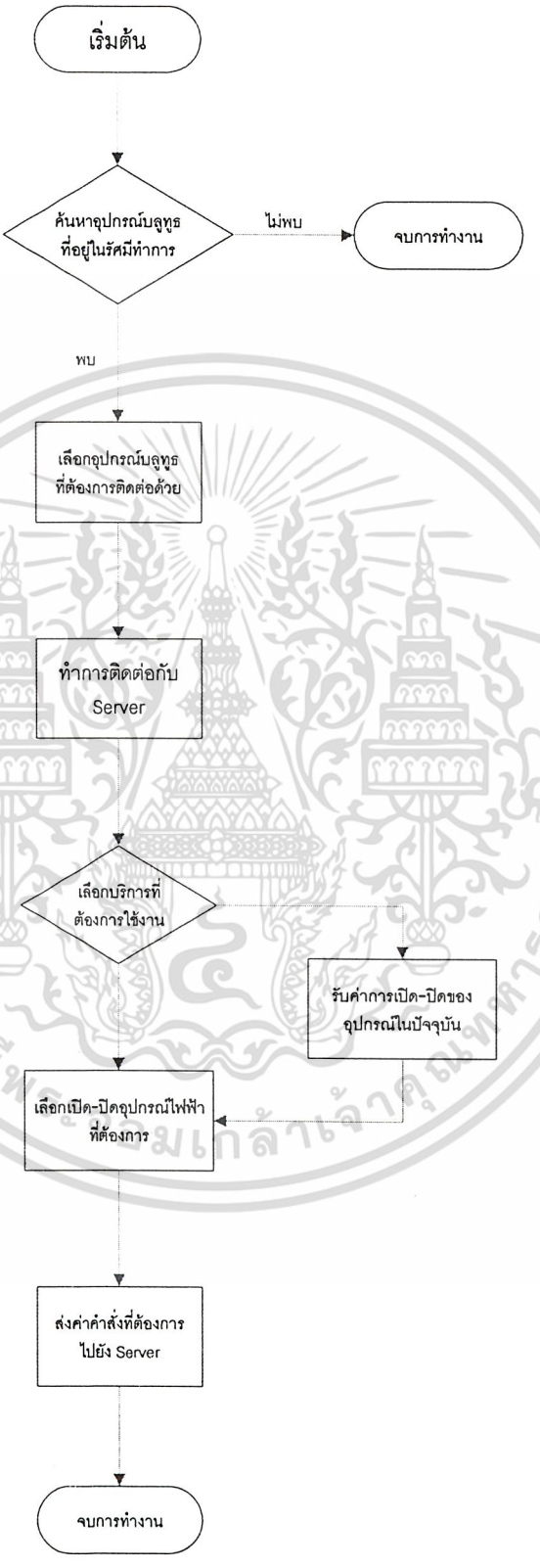
3. ส่วนการส่งรหัสคำสั่ง

ส่วนนี้จะทำงานเมื่อผู้ใช้กดส่งคำสั่งไปยังส่วนโปรแกรมบนเครื่องคอมพิวเตอร์ รหัสคำสั่งที่เป็น string จะถูกส่งมาจากส่วนเลือกการเปิด-ปิดอุปกรณ์ไฟฟ้าที่หน้าจอ interface รหัสคำสั่งที่รับมานี้จะถูกส่งเป็นชุดคำสั่งออกไปผ่านทาง protocol L2CAP

3.1.3 ส่วนการเลือกรหัสคำสั่ง

ส่วนนี้เป็นส่วนที่ใช้ในการค้นหารหัสคำสั่งที่ผู้ใช้เลือกที่จะเปิด-ปิดเครื่องใช้ไฟฟ้าต่างๆ บนหน้าจอ interface ในการค้นหารหัสคำสั่งจะทำการตรวจสอบชนิดของเครื่องใช้ไฟฟ้าก่อนแล้วจึงทำการเปรียบเทียบคำสั่งที่รับมาว่าเป็นการเปิดหรือปิดเครื่องใช้ไฟฟ้าชนิดนั้น และนำคำสั่งที่ได้ส่งไปยังส่วนการส่งรหัสคำสั่งเพื่อส่งต่อไปยังโปรแกรมบนเครื่องคอมพิวเตอร์ และถ้าผู้ใช้ต้องการทราบสถานะข้อมูลการเปิด-ปิดของอุปกรณ์ไฟฟ้าในห้องจำลองที่ใช้งานอยู่ในปัจจุบันก็สามารถกดเลือกคำสั่งได้จากหน้าจอ interface บนโทรศัพท์มือถือ เมื่อคำสั่งถูกส่งออกไปยังส่วนโปรแกรมบนเครื่องคอมพิวเตอร์ผ่านส่วนการส่งรหัสคำสั่ง โปรแกรมบนเครื่องคอมพิวเตอร์ก็จะทำการประมวลผลคำสั่งและรับข้อมูลดังกล่าวมาจาก MCS-51 เพื่อแสดงผลให้ผู้ใช้ทราบที่หน้าจอ interface บนโทรศัพท์มือถือต่อไป

โดยการทำงานของโปรแกรมบนโทรศัพท์มือถือจะมีโฟลว์ชาร์ตการทำงานดังรูปที่ 3.2



รูปที่ 3.2 โฟลว์ชาร์ตแสดงการทำงานของโปรแกรมบนโทรศัพท์มือถือ

3.2 ส่วนโปรแกรมบนเครื่องคอมพิวเตอร์

ส่วนโปรแกรมบนเครื่องคอมพิวเตอร์ได้ถูกเขียนขึ้นด้วยภาษา Java โดยถูกพัฒนาบนโปรแกรม JCreator ส่วนโปรแกรมบนเครื่องคอมพิวเตอร์นี้ได้แบ่งการทำงานออกเป็น 2 ส่วน คือ ส่วนเชื่อมต่อด้วยบลูทูธ และ ส่วนเชื่อมต่อด้วย RS-232

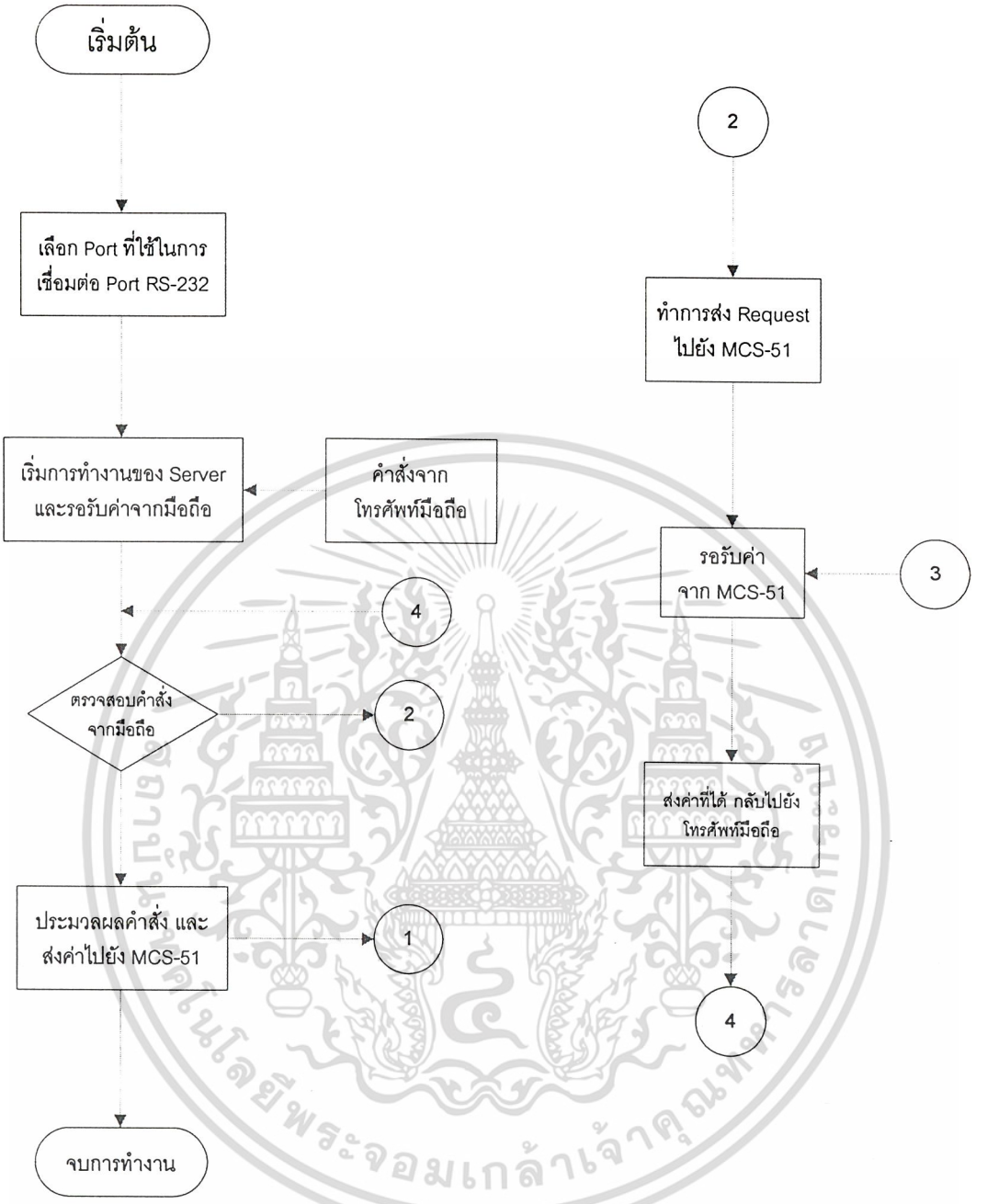
3.2.1 ส่วนเชื่อมต่อด้วยบลูทูธ

ส่วนนี้จะทำงานโดยการสร้ างบริการขึ้นมาซึ่งมี UUID เป็น 1122334455 โดยใช้ protocol L2CAP การทำงานส่วนนี้จะมีลักษณะเป็น server ซึ่งจะทำการเปิดการเชื่อมต่อรอไว้เพื่อให้โทรศัพท์มือถือซึ่งมีลักษณะการทำงานเป็น client ติดต่อมา ข้อมูลที่รับเข้าจากโทรศัพท์มือถือจะมีลักษณะเป็นชุดคำสั่งซึ่งเป็นรหัส binary จะถูกนำมาประมวลผล แล้วส่งต่อไปสั่งงาน Microcontroller MCS-51 เพื่อทำการตัดต่อกระแสไฟฟ้าในห้องจำลองผ่านทางพอร์ตอนุกรม RS-232

3.2.2 ส่วนเชื่อมต่อด้วย RS-232

ส่วนนี้จะเป็นส่วนที่เชื่อมต่อโปรแกรมบนเครื่องคอมพิวเตอร์กับวงจรตัดต่อกระแสไฟฟ้าในห้องจำลอง โดยจะส่งคำสั่งที่รับมาจากส่วนการเชื่อมต่อด้วยบลูทูธไปให้กับ MCS-51 เพื่อทำการประมวลผลคำสั่งต่อไป และยังสามารถรับข้อมูลการเปิด-ปิดของอุปกรณ์ไฟฟ้าในห้องจำลองที่ใช้งานอยู่ในปัจจุบันจาก MCS-51 เพื่อแสดงผลทางหน้าจอ interface บนโทรศัพท์มือถือผ่านทาง protocol L2CAP ได้อีกด้วย

โดยการทำงานของโปรแกรม Server บนเครื่องคอมพิวเตอร์จะมีโฟลว์ชาร์ตการทำงานดังรูปที่ 3.3



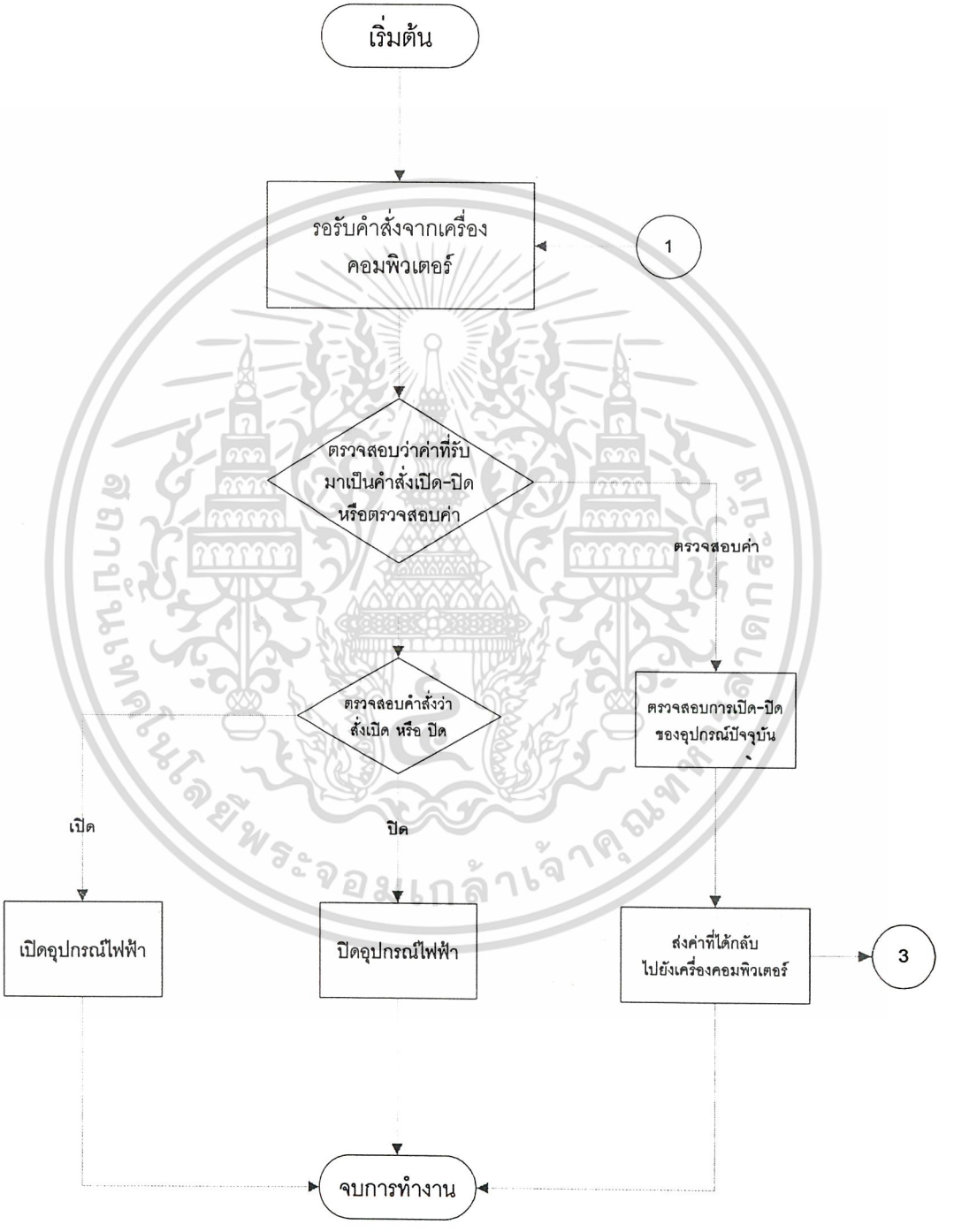
รูปที่ 3.3 โฟลว์ชาร์ตแสดงการทำงานของโปรแกรม Server

3.3 ส่วนวงจรตัดต่อกระแสไฟฟ้าในห้องจำลอง

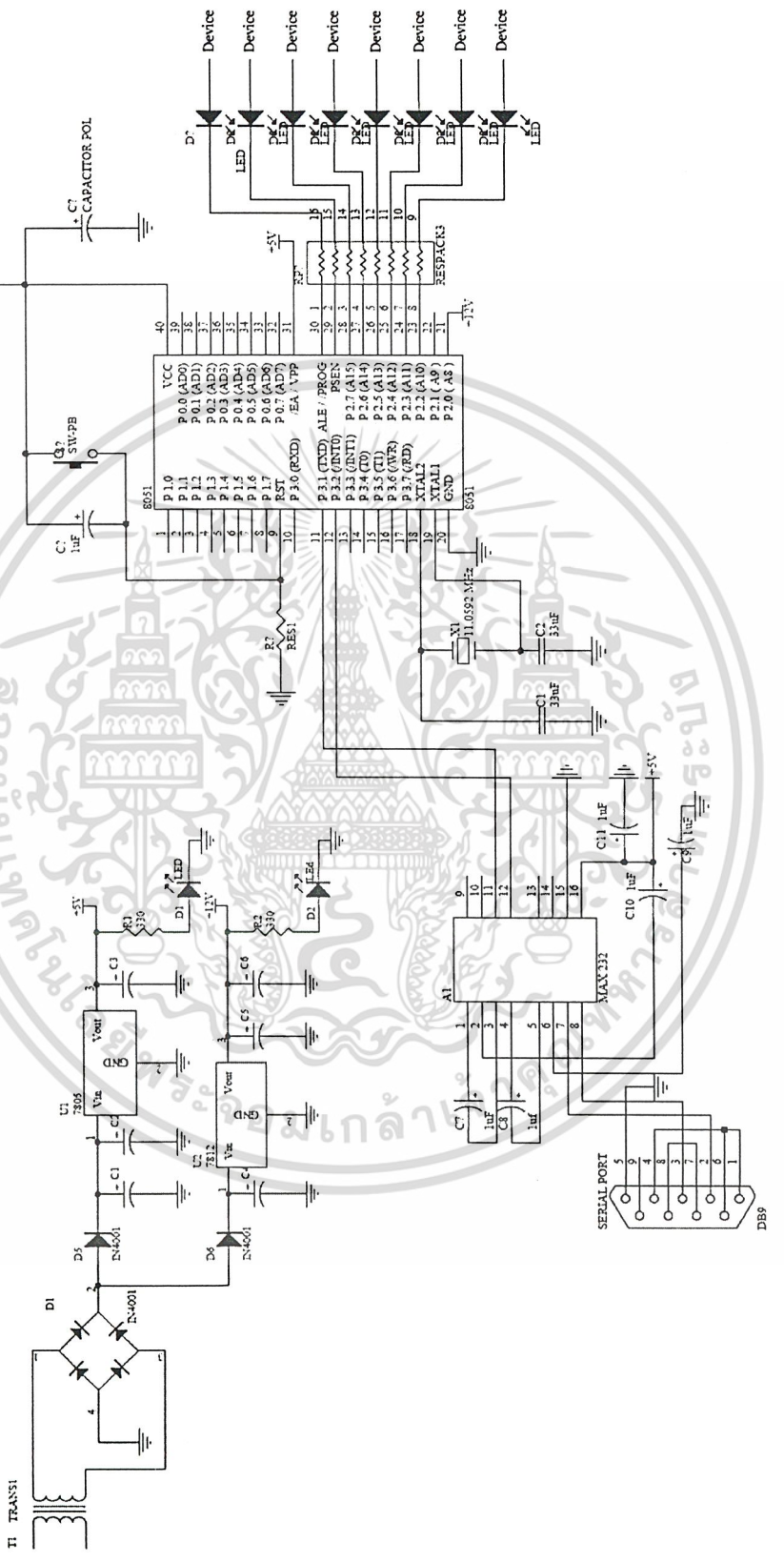
ส่วนนี้ได้ทำการติดต่อกับเครื่องคอมพิวเตอร์ผ่านพอร์ตอนุกรม RS-232 โดยทำการรับ-ส่งข้อมูลกับเครื่องคอมพิวเตอร์ ในส่วนของการรับข้อมูลจากเครื่องคอมพิวเตอร์ ไมโครคอนโทรลเลอร์ MCS-51 จะนำข้อมูลที่รับมาประมวลผลและส่งเป็นคำสั่งไปควบคุมการเปิด-ปิดอุปกรณ์ไฟฟ้าที่ห้องควบคุมควบคุมอุปกรณ์ไฟฟ้าจำลอง และในส่วนการส่งข้อมูลไปยังเครื่องคอมพิวเตอร์ก็จะนำข้อมูลการเปิด-ปิดอุปกรณ์ไฟฟ้าในห้องจำลองที่ใช้งานอยู่ในปัจจุบันจากไมโครคอนโทรลเลอร์ MCS-51 ส่งไปยังเครื่องคอมพิวเตอร์เพื่อแสดงผลยังหน้าจอโทรศัพท์มือถือต่อไป

ในส่วนของไมโครคอนโทรลเลอร์ MCS-51 ได้ใช้ CPU รุ่น AT89C51 ของบริษัท ATMEL เนื่องจากมีราคาถูกและมีประสิทธิภาพดี และได้ใช้ภาษา C ในการพัฒนาโปรแกรม

โดยการทำงานของโปรแกรมบนไมโครคอนโทรลเลอร์ MCS-51 จะมีโฟลว์ชาร์ตการทำงานดังรูปที่ 3.4 และมีรูปวงจรดังรูปที่ 3.5



รูปที่ 3.4 โฟลว์ชาร์ตแสดงการทำงานของโปรแกรมบน MCS-51



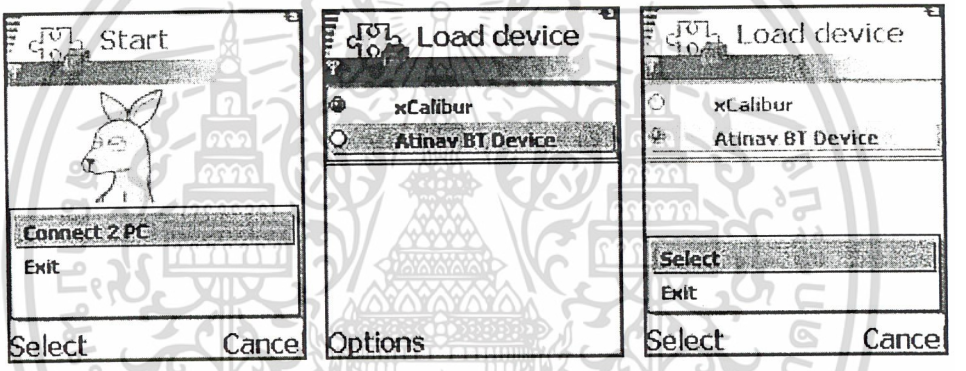
รูปที่ 3.5 วงจรตัดต่อกระแสไฟฟ้าในห้องจำลอง

บทที่ 4

การทำงานของโปรแกรม

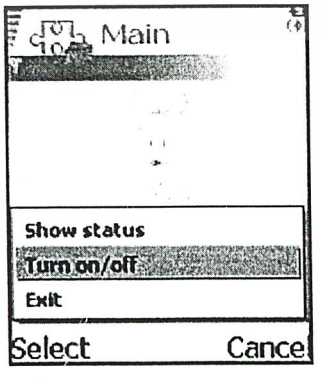
4.1 ส่วนโปรแกรมควบคุมเครื่องใช้ไฟฟ้าบนมือถือ

เมื่อผู้ใช้เริ่มต้นการทำงานของโปรแกรมผู้ใช้จะต้องเลือกไปที่ Options เพื่อทำการเลือกว่าจะติดต่อกับอุปกรณ์บลูทูธ (Connect 2 PC) หรือจะออกจากโปรแกรม (Exit) ถ้าผู้ใช้เลือก Exit ก็จะออกจากโปรแกรม แต่ถ้าเลือก Connect 2 PC โปรแกรมจะเริ่มต้นการค้นหาอุปกรณ์บลูทูธที่อยู่ในรัศมีทำการทุกตัว และจะแสดงอุปกรณ์ที่พบให้ผู้ใช้เลือกว่าจะทำการติดต่อกับอุปกรณ์ใด แสดงดังรูปที่ 4.1 หลังจากที่ผู้ใช้เลือกอุปกรณ์บลูทูธที่จะทำการเชื่อมต่อ ระบบก็จะทำการค้นหาบริการซึ่งมี UUID = 1122334455 ผลที่ได้จากการค้นหาบริการคือ Service URL



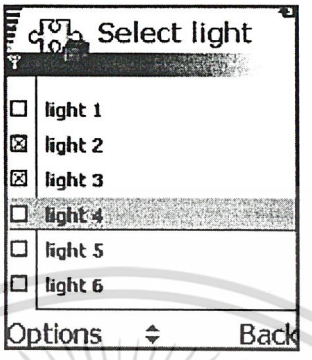
รูปที่ 4.1 หน้าจอ interface ขณะเริ่มต้นการใช้งานโปรแกรมและ
ผลการค้นหาอุปกรณ์บลูทูธที่จะทำการเชื่อมต่อ

หลังจากที่เชื่อมต่อกันได้แล้วผู้ใช้กดเลือกไปที่ Options ก็จะปรากฏหน้าจอ interface ให้ผู้ใช้เลือกว่าจะทำการแสดงสถานะของอุปกรณ์ไฟฟ้าที่ใช้งานอยู่ในปัจจุบันที่ห้องจำลอง (Show status) หรือจะทำการส่งคำสั่งการเปิด-ปิดอุปกรณ์ไฟฟ้า (Turn on/off) ดังแสดงในรูปที่ 4.2



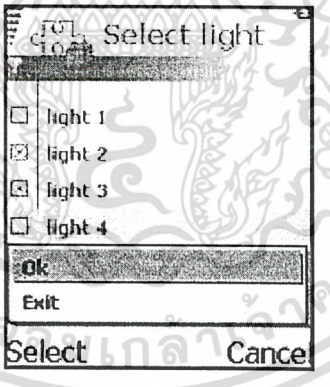
รูปที่ 4.2 หน้าจอ interface ที่ให้ผู้ใช้เลือกระหว่าง Show status กับ Turn on/off

ถ้าผู้ใช้เลือกคำสั่ง Show status โปรแกรมจะทำการส่งคำสั่งไปยังคอมพิวเตอร์ให้ทำการประมวลผลคำสั่ง หลังจากนั้นข้อมูลการเปิด-ปิดของอุปกรณ์ไฟฟ้าในห้องจำลองที่ใช้งานอยู่ในปัจจุบันก็จะถูกส่งมาแสดงผลยังหน้าจอ interface ที่โทรศัพท์มือถือ ดังรูปที่ 4.3



รูปที่ 4.3 หน้าจอ interface แสดงสถานะการเปิด-ปิดอุปกรณ์ไฟฟ้าในห้องจำลองที่ใช้งานอยู่ในปัจจุบัน

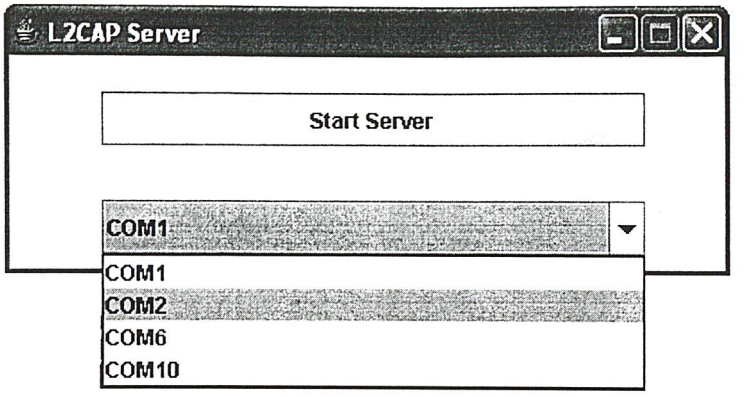
แต่ถ้าผู้ใช้เลือกคำสั่ง Turn on/off โปรแกรมจะทำการแสดงหน้าจอ interface ให้ผู้ใช้เลือกเปิด-ปิดอุปกรณ์ไฟฟ้าที่ห้องจำลองตามต้องการ เมื่อผู้ใช้เลือกเสร็จเรียบร้อยแล้วก็ทำการกดส่งคำสั่งออกไป ดังรูปที่ 4.4 โปรแกรมก็จะทำการค้นหารหัสคำสั่งแล้วส่งไปยังส่วนโปรแกรมบนเครื่องคอมพิวเตอร์



รูปที่ 4.4 หน้าจอ interface ตามขั้นตอนต่างๆเมื่อผู้ใช้เลือกคำสั่ง Turn on/off

4.2 ส่วนโปรแกรมบนเครื่องคอมพิวเตอร์

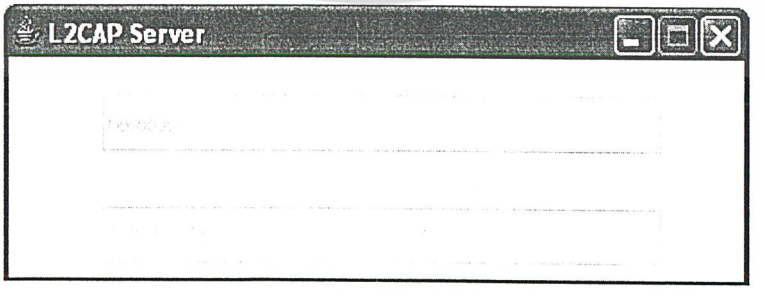
เริ่มต้นผู้ใช้ต้องทำการเลือก comport ว่าพอร์ท RS-232 นั้นใช้งาน comport ใดอยู่เพื่อใช้ในการส่งข้อมูลไปยังอุปกรณ์ hardware หลังจากนั้นก็ทำการเปิดการทำงานของ server ดังรูปที่ 4.5 โดยตัว server จะทำการส่งสัญญาณออกไปรอบๆเพื่อรอรับการติดต่อจาก client ที่อยู่ในรัศมีทำการ



รูปที่ 4.5 หน้าจอ interface ที่คอมพิวเตอร์แสดงการเลือก comport และการเปิดการทำงานของ server

เมื่อ server ได้รับการติดต่อจาก client แล้วก็จะทำการเชื่อมต่อกัน โดยทำการตรวจสอบ UUID ก่อนว่ามีค่าตรงกันหรือไม่ แล้วถ้า client จะทำการส่งข้อมูลการเปิด-ปิดอุปกรณ์ไฟฟ้ามาให้ server ข้อมูลก็จะถูกประมวลผลแล้วนำไปเปิด-ปิดอุปกรณ์ไฟฟ้าที่ห้องจำลองตามคำสั่งที่ได้รับมา และถ้า client ต้องการทราบสถานะข้อมูลการเปิด-ปิดของอุปกรณ์ไฟฟ้าในห้องจำลองที่ใช้งานอยู่ในปัจจุบัน ข้อมูลดังกล่าวก็จะถูกส่งกลับไปให้ client ตามต้องการ โดยข้อมูลการรับและส่งนี้ยังสามารถแสดงได้ที่หน้าจอคอมพิวเตอร์ดังรูปที่ 4.6 และรูปที่ 4.7

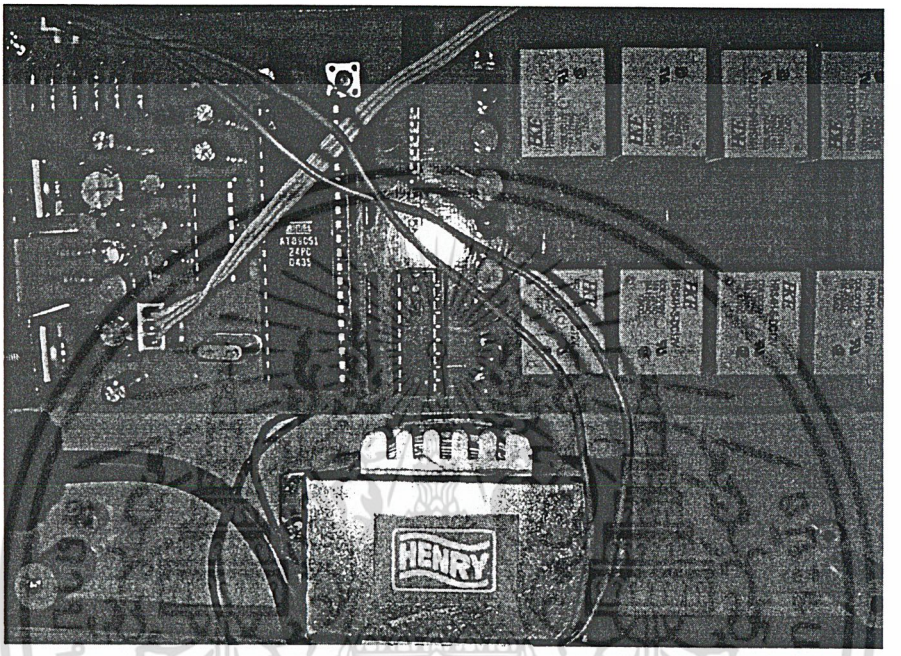
รูปที่ 4.6 หน้าจอ interface ที่คอมพิวเตอร์แสดงข้อมูลที่รับจากเครื่อง client



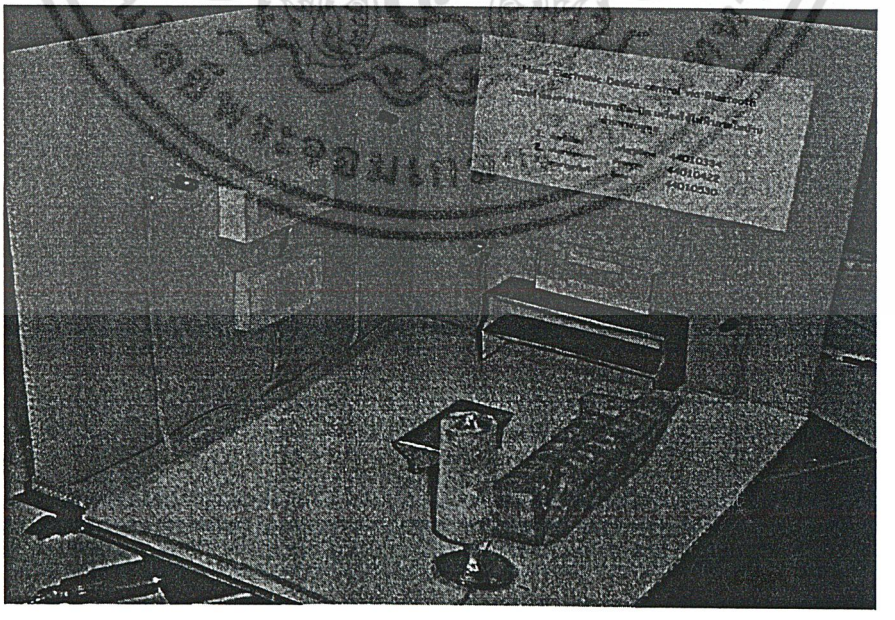
รูปที่ 4.7 หน้าจอ interface ที่คอมพิวเตอร์แสดงข้อมูลส่งกลับไปที่เครื่อง client

4.3 ส่วนวงจรตัดต่อกระแสไฟฟ้าในห้องจำลอง

ส่วนนี้จะทำการจำลองการเปิด-ปิดอุปกรณ์ไฟฟ้าที่ห้องจำลองตามคำสั่งที่ได้รับมาจากโทรศัพท์มือถือ (client) ซึ่งจะประมวลผลคำสั่งที่ MCS-51 และจะแสดงผลการเปิด-ปิดอุปกรณ์ผ่านทางหลอดไฟ LED จำนวน 8 ดวงโดยมีรีเลย์เป็นตัวตัดต่อกระแสไฟฟ้า ส่วนวงจรตัดต่อกระแสไฟฟ้านี้มีวงจรดังแสดงได้ดังรูปที่ 4.8 และรูปที่ 4.9 แสดงห้องจำลองที่แทนอุปกรณ์ไฟฟ้าทุกตัวด้วยหลอดไฟ LED



รูปที่ 4.8 วงจรรวมส่วนตัดต่อกระแสไฟฟ้าในห้องจำลอง



รูปที่ 4.9 ห้องจำลองอุปกรณ์ไฟฟ้า

บทที่ 5

บทวิจารณ์และบทสรุป

5.1 ข้อสรุปและข้อเสนอแนะ

ระบบจำลองการควบคุมการเปิด-ปิดเครื่องใช้ไฟฟ้าภายในบ้านผ่านทางบลูทูธได้มีการพัฒนาโปรแกรมบนโทรศัพท์มือถือเพื่อใช้ในการเปิด-ปิดอุปกรณ์ไฟฟ้าในบ้านและพัฒนาโปรแกรมบนเครื่อง desktop โดยมีวัตถุประสงค์เพื่อทดสอบการทำงานของมาตรฐาน JSR-82 Java Bluetooth API ซึ่งถูกนำมาใช้งานทั้งบนเครื่องโทรศัพท์มือถือและบนเครื่อง desktop ในการใช้งานผู้ใช้จะสั่งเปิดโปรแกรมควบคุมการเปิด-ปิดอุปกรณ์ไฟฟ้าบนเครื่องโทรศัพท์มือถือเพื่อทำการค้นหาอุปกรณ์บลูทูธและเลือกเปิด-ปิดอุปกรณ์ไฟฟ้าผ่านโทรศัพท์มือถือ คำสั่งจะถูกส่งผ่านเครือข่ายบลูทูธจากโทรศัพท์มือถือมายังเครื่องคอมพิวเตอร์ server ที่จะทำการแปลคำสั่งและแสดงผลของคำสั่งที่ได้รับออกมายังหน้าจอคอมพิวเตอร์ และส่งคำสั่งที่ได้ออกไปยังพอร์ตอนุกรม RS-232 ให้ MCS-51 รับคำสั่งไปเปิด-ปิดอุปกรณ์ไฟฟ้าที่ห้องจำลองต่อไป ระบบจำลองการควบคุมการเปิด-ปิดเครื่องใช้ไฟฟ้าภายในบ้านผ่านทางบลูทูธนี้เป็นเพียงการประยุกต์ใช้งานเบื้องต้นของการควบคุมอุปกรณ์ไฟฟ้าต่างๆผ่านเทคโนโลยีบลูทูธ ซึ่งอาจสามารถนำไปประยุกต์ใช้กับระบบการควบคุมอื่นๆได้อย่างมีประสิทธิภาพมากยิ่งขึ้น

5.2 แนวทางการประยุกต์และพัฒนาต่อ

นอกจากการใช้งานการเปิด-ปิดอุปกรณ์ไฟฟ้าผ่านทางบลูทูธแล้วในอนาคตยังสามารถพัฒนาให้สามารถควบคุมอุปกรณ์เครื่องใช้ไฟฟ้าต่างๆ เช่น สามารถสั่งเปิด-ปิด หรือ เปลี่ยนช่องโทรทัศน์ได้ผ่านบลูทูธของโทรศัพท์มือถือ เป็นต้น และยังสามารถใช้กับระบบโทรทัศน์วงจรปิดให้สามารถสั่งเปิด-ปิด,ดูภาพหรือตรวจสอบภาพได้จากโทรศัพท์มือถือในระบบรักษาความปลอดภัย หรือพัฒนาเพื่อประยุกต์ใช้กับบุคคลที่ไม่สามารถเคลื่อนไหวร่างกายได้สะดวกให้สามารถสั่งการอุปกรณ์ไฟฟ้าได้ง่ายขึ้นอีกด้วย

ภาคผนวก

Source Code

ของโปรแกรมบนโทรศัพท์มือถือ

ภาคผนวก

Source Code

ของโปรแกรม Server บนคอมพิวเตอร์

```
import javax.bluetooth.*;
import java.io.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class L2CAPServer extends JFrame implements
BTCallback,ComportCallback,ActionListener {
      private static final String UUID_STRING = "1122334455";
      private static final String UUID_SERVICE = "L2CAPchat";
    private JTextField incoming = new JTextField();
    private JTextField outgoing = new JTextField();
      private JButton startCommand = new JButton("Start Server");
      private JComboBox SelectComport;
      private BTServicAndDeviceAgent BAgent;
      private Container contentPane;

      private ComportAgent ComportAgent;
      private boolean BlueToothBlock = true;
      private String tmp = "";
      private String portname = "COM1";

      public L2CAPServer() {
            String[] comportname = {"COM1","COM2","COM6","COM10"};
            BAgent = new
BTServicAndDeviceAgent(UUID_STRING,UUID_SERVICE,this);
            SelectComport = new JComboBox(comportname);
        contentPane = getContentPane();
        contentPane.setLayout(null);
        setResizable(false);

        incoming.setEnabled(false);
        incoming.setSize(300,30);
        incoming.setLocation(50,20);

        outgoing.setEnabled(false);
        outgoing.setSize(300,30);
        outgoing.setLocation(50,80);
        outgoing.addActionListener(this);

        startCommand.setLocation(50,20);
        startCommand.setSize(300,30);
        startCommand.addActionListener(this);

        SelectComport.setSize(300,30);
        SelectComport.setLocation(50,80);
        SelectComport.addActionListener(this);

        contentPane.removeAll();
      contentPane.add(startCommand);
      contentPane.add(SelectComport);

        addWindowListener(new WindowAdapter(){
                  public void windowClosing(WindowEvent e) {
                        try{
                              BAgent.StopServer();
                              ComportAgent.StopAgent();

                              BAgent.ReleaseResource();
                              BAgent = null;
                              ComportAgent = null;
```

```
                        System.exit(0);
                    }catch(Exception ex){
                        BAgent = null;
                        ComportAgent = null;
                        System.exit(0);
                    }
                }
        });
}

private void StartServer(){
  BAgent.StartServer();
  ComportAgent = new ComportAgent(portname,this);
  contentPane.removeAll();
  contentPane.add(incoming);
    contentPane.add(outgoing);
    repaint();
}

public void actionPerformed(ActionEvent e){
  if(e.getSource() == outgoing){
        BAgent.ToSendData(outgoing.getText());
  }else if(e.getSource() == startCommand) {
        StartServer();
  }else if(e.getSource() == SelectComport){
        portname = (String)SelectComport.getSelectedItem();
  }
}

  public static void main(String[] argv) {
        L2CAPServer frame = new L2CAPServer();
        frame.setSize(400,150);
        frame.setTitle("L2CAP Server");
        frame.setVisible(true);
  }

  public void ComportReciveData(String data)
  {
        BlueToothBlock = false;
        System.out.println(data);
        tmp = data;
  }

  public void ReciveData(String message) {
        incoming.setText(message);
        if(message.equalsIgnoreCase("rxxxxxxx"))
        {
              String buff = "";
              ComportAgent.SendData("r");
              BlueToothBlock = true;
              while(BlueToothBlock);
              byte tmp1 = Byte.parseByte(tmp);

              for(int i=0;i<8;i++){
                    if((tmp1 & 0x80) == 0x80){
                          buff += "1";
                    }else{
                          buff += "0";
                    }
                    tmp1 <<=1;
              }
```

```
        outgoing.setText(buff);
        BAgent.ToSendData(buff);
    }else{
        if(message.charAt(0) == '1'){
            ComportAgent.SendData("1");
            BlueToothBlock = true;
        }else{
            ComportAgent.SendData("a");

            BlueToothBlock = true;
        }
        while(BlueToothBlock);

        if(message.charAt(1) == '1'){
            ComportAgent.SendData("2");
            BlueToothBlock = true;
        }else{
            ComportAgent.SendData("b");

            BlueToothBlock = true;
        }
        while(BlueToothBlock);

        if(message.charAt(2) == '1'){
            ComportAgent.SendData("3");

            BlueToothBlock = true;
        }else{
            ComportAgent.SendData("c");

            BlueToothBlock = true;
        }
        while(BlueToothBlock);

        if(message.charAt(3) == '1'){
            ComportAgent.SendData("4");
            BlueToothBlock = true;
        }else{
            ComportAgent.SendData("d");
            BlueToothBlock = true;
        }
        while(BlueToothBlock);

        if(message.charAt(4) == '1'){
            ComportAgent.SendData("5");
            BlueToothBlock = true;
        }else{
            ComportAgent.SendData("e");
            BlueToothBlock = true;
        }
        while(BlueToothBlock);

        if(message.charAt(5) == '1'){
            ComportAgent.SendData("6");
            BlueToothBlock = true;
        }else{
            ComportAgent.SendData("f");
            BlueToothBlock = true;
        }
        while(BlueToothBlock);
```

```
                    if(message.charAt(6) == '1'){
                          ComportAgent.SendData("7");
                          BlueToothBlock = true;
                    }else{
                          ComportAgent.SendData("g");
                          BlueToothBlock = true;
                    }
                    while(BlueToothBlock);

                    if(message.charAt(7) == '1'){
                          ComportAgent.SendData("8");
                          BlueToothBlock = true;
                    }else{
                          ComportAgent.SendData("h");
                          BlueToothBlock = true;
                    }
                    while(BlueToothBlock);
              }
        }
        public void FoundDevice(String[] devices) {}
}
```

```
import java.io.*;
import java.util.*;
import javax.comm.*;

public class ComportAgent implements Runnable,
SerialPortEventListener{
      private String portName;
      private ComportCallback callback;
      private boolean running = false;
      private Thread runningThread;
      private CommPortIdentifier portId;
    private Enumeration portList;

    private InputStream inputStream;
    SerialPort serialPort;
      private OutputStream outputStream;
      private ComportAgent(){
      }

      public ComportAgent(String portName,ComportCallback callback){
            this.callback = callback;
            this.portName = portName;
            running = true;
            portList = CommPortIdentifier.getPortIdentifiers();

        while (portList.hasMoreElements()) {
            portId = (CommPortIdentifier) portList.nextElement();
            if (portId.getPortType() ==
CommPortIdentifier.PORT_SERIAL) {
                if (portId.getName().equals(portName)) {
                    try {
                        serialPort = (SerialPort)
                            portId.open("SimpleKrisMTComportAgent",
2000);
                    } catch (PortInUseException e)
{e.printStackTrace();}
                    try {
      inputStream  = serialPort.getInputStream();
                        outputStream = serialPort.getOutputStream();
                    } catch (IOException e) {e.printStackTrace();}

                    try {
                    serialPort.addEventListener(this);
                                } catch (TooManyListenersException e)
{}
                    serialPort.notifyOnDataAvailable(true);

                      try {
                          serialPort.setSerialPortParams(9600,
                              SerialPort.DATABITS_8,
                              SerialPort.STOPBITS_1,
                              SerialPort.PARITY_NONE);
                      } catch (UnsupportedCommOperationException e)
{e.printStackTrace();}
                      runningThread = new Thread(this);
                      runningThread.start();
                }
            }
        }
      }
      public boolean SendData(String data){
```

```
            try {
      outputStream.write(data.getBytes());
        } catch (IOException e) { e.printStackTrace();}
        try{
      this.wait(2000);
        }catch(Exception e){};
        return true;
      }

      public void StopAgent(){
            running = false;
            runningThread.stop();
      }

      public void run() {
      while(running){
            try{
                  runningThread.sleep(2000);
            }catch(InterruptedException e){ e.printStackTrace();}
      }
    }

    public void serialEvent(SerialPortEvent event) {
        switch(event.getEventType()) {
        case SerialPortEvent.BI:
        case SerialPortEvent.OE:
        case SerialPortEvent.FE:
        case SerialPortEvent.PE:
        case SerialPortEvent.CD:
        case SerialPortEvent.CTS:
        case SerialPortEvent.DSR:
        case SerialPortEvent.RI:
        case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
            break;
        case SerialPortEvent.DATA_AVAILABLE:
            byte[] readBuffer = new byte[1];
            try {
                while (inputStream.available() > 0){
                    int numBytes = inputStream.read(readBuffer);
                }
                callback.ComportReciveData(new String(readBuffer));
            } catch (IOException e) {e.printStackTrace();}
            break;
        }
    }
}
```

```
import javax.bluetooth.*;
import java.util.*;
import java.io.*;

public final class BTServicAndDeviceDiscovery implements
DiscoveryListener {
      private Vector remoteDevices = new Vector();
      private ServiceRecord serviceRecord;
      private BTServicAndDeviceAgent callback;
      private String SERVICE_NAME;
      public BTServicAndDeviceDiscovery(String
SERVICE_NAME,BTServicAndDeviceAgent callback) {
            this.SERVICE_NAME = SERVICE_NAME;
            this.callback = callback;
      }

      public void servicesDiscovered(int transID, ServiceRecord[]
servRecord) {
            for(int i = 0; i < servRecord.length; i++) {
            DataElement serviceNameElement =
servRecord[i].getAttributeValue(0X0100);
            String serviceName =
(String)serviceNameElement.getValue();
            if(serviceName.equals(SERVICE_NAME)){
                serviceRecord = servRecord[0];
            }
        }
      }

    public void serviceSearchCompleted(int transID, int respCode) {
            String message = null;
        if (respCode ==
DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE) {
            message = "Device not reachable";
        }
        else if (respCode ==
DiscoveryListener.SERVICE_SEARCH_NO_RECORDS) {
            message = "Service not available";
        }
        else if (respCode ==
DiscoveryListener.SERVICE_SEARCH_COMPLETED) {
            message = "Service search completed";
        }
        else if (respCode ==
DiscoveryListener.SERVICE_SEARCH_TERMINATED) {
            message = "Service search terminated";
        }
        else if (respCode == DiscoveryListener.SERVICE_SEARCH_ERROR)
{
            message = "Service search error";
        }
        callback.serviceSearchFinished(serviceRecord, message);
      }
      public void inquiryCompleted(int discType) {
            String message = null;

        if (discType == INQUIRY_COMPLETED) {
            message = "Inquiry completed";
        } else if (discType == INQUIRY_TERMINATED) {
            message = "Inquiry terminated";
```

```
        } else if (discType == INQUIRY_ERROR) {
            message = "Inquiry error";
        }
        RemoteDevice[] devices = new
RemoteDevice[remoteDevices.size()];
        for(int i = 0; i < remoteDevices.size(); i++) {
            devices[i] = (RemoteDevice)remoteDevices.elementAt(i);
        }
            callback.deviceInquiryFinished(devices, message);
      }
    public void deviceDiscovered(RemoteDevice btDevice, DeviceClass
cod) {
            remoteDevices.addElement(btDevice);
      }
}
```

```
import java.io.*;
import javax.bluetooth.*;
import com.atinav.standardedition.io.*;

public final class BTServicAndDeviceAgent
{
      private String UUID_STRING;
      private String SERVICE_NAME;
      private LocalDevice localDevice;
    private DiscoveryAgent agent;
    private L2CAPConnection conn;
    private RemoteDevice[] remoteDevices;
      private boolean running = false;
      private BTCallback callback;
      private String msgout;
      private String msgin;
      private BTServicAndDeviceDiscovery discoverer;
      private String[] devices;
      private BTServicAndDeviceAgent(){
      }

      public BTServicAndDeviceAgent(String UUID_STRING,String
SERVICE_NAME,BTCallback callback){
            this.UUID_STRING = UUID_STRING;
            this.SERVICE_NAME = SERVICE_NAME;
            this.callback = callback;
            discoverer = new
BTServicAndDeviceDiscovery(SERVICE_NAME,this);
            try{
            localDevice = LocalDevice.getLocalDevice();
            agent = localDevice.getDiscoveryAgent();
        }catch(BluetoothStateException bse){
                  bse.printStackTrace();
        }
      }
      public void StartServer() {
            new Thread() {
                  public void run() {
                        try{

      localDevice.setDiscoverable(DiscoveryAgent.GIAC);
                              L2CAPConnectionNotifier notifier =
(L2CAPConnectionNotifier)Connector.open("btl2cap://localhost:" +
UUID_STRING + ";name=" + SERVICE_NAME);
                              ServiceRecord record =
localDevice.getRecord(notifier);
                              String connURL =
record.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT,
false);
                              conn = notifier.acceptAndOpen();
                              new Thread(){
                                    public void run(){
                                          running = true;
                                          ToReciveData();
                                    }
                              }.start();
                        } catch(IOException ioe) {
                              ioe.printStackTrace();
                        }
                  }
            }.start();
```

```
    }

    public void StopServer() {
          running = false;
    }

    public void SearchDevice() {
                try {
               agent.startInquiry(DiscoveryAgent.GIAC, discoverer);

          } catch(BluetoothStateException bse){
             bse.printStackTrace();
          }
    }
    public void SearchService(int index) {
          int[] attrSet = {0x0100};//return service name attribute
          UUID[] uuidSet = new UUID[1];
          uuidSet[0]=new UUID(UUID_STRING, false);
          try{
                agent.searchServices(attrSet, uuidSet,
remoteDevices[index], discoverer);
          } catch(BluetoothStateException bse){
                bse.printStackTrace();
       }
    }

    public void StopClient() {
          running = false;
    }

    public void deviceInquiryFinished(RemoteDevice[] remoteDevices,
String message){
          this.remoteDevices = remoteDevices;
          devices = new String[remoteDevices.length];
       for(int i = 0; i < remoteDevices.length; i++) {
          try {
                String name =
remoteDevices[i].getFriendlyName(false);
                devices[i] = name;
          }catch(IOException ioe){
                ioe.printStackTrace();
          }
       }
       callback.FoundDevice(devices);
    }
    public void serviceSearchFinished(ServiceRecord serviceRecord,
String message){
          String url =
serviceRecord.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT
, false);
       try{
          conn = (L2CAPConnection)Connector.open(url);
                new Thread(){
             public void run(){
                         running = true;
                ToReciveData();
             }
                }.start();
          }catch(IOException ioe) {
                ioe.printStackTrace();
```

```
        }
    }

    public void ToReciveData() {
        while(running){
        try{
            if(conn.ready()){
                int receiveMTU = conn.getReceiveMTU();
                byte[] data = new byte[receiveMTU];
                int length = conn.receive(data);
                msgin = new String(data, 0, length);
                new Thread() {
    public void run() {
        callback.ReciveData(msgin);
    }
                }.start();
            }
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
      }
    }


    public void ToSendData(String message) {
        this.msgout = message;
        new Thread(){
            public void run(){
                try {
                                byte[] data = msgout.getBytes();

                                int transmitMTU =
conn.getTransmitMTU();
                                if(data.length <= transmitMTU){
                                    conn.send(data);
                                }else {
                                    System.out.println("Message
too long!");
                                }
                            } catch(IOException ioe) {
                                ioe.printStackTrace();
                            }
                        }
                }.start();
    }

    public void ReleaseResource() {
        try{
            if(localDevice != null) localDevice = null;
            if(agent != null) agent = null;
            if(remoteDevices != null) remoteDevices = null;
        if(conn != null) conn.close();
      }catch(IOException ioe){
        ioe.printStackTrace();
      }
    }
}
```

Source Code
ของโปรแกรมบนโทรศัพท์มือถือ

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import javax.bluetooth.*;
import java.io.*;

public class Mobile extends MIDlet implements
BTCallback,CommandListener {
   private Form f_main = new Form("Main");
   private Form f_status = new Form("Status");
   private Form f_start = new Form("Start");
   private Form f_search = new Form("Search Device");
   private Form f_loaddevice = new Form("Load device");
   private Command c_connect = new Command("Connect 2
PC",Command.ITEM,1);
   private Command c_start = new Command("Start
server",Command.ITEM,1);
   private Command c_status = new Command("Show status",
Command.ITEM, 1);
   private Command c_on = new Command("Turn on/off", Command.ITEM, 1);
   private Command c_back = new Command("Back", Command.BACK,1);
   private Command c_exit = new Command("Exit", Command.EXIT,1);
   private Command c_ok = new Command("Ok",Command.OK,1);
   private Command c_select = new Command("Select", Command.ITEM, 1);
   private Image i_logo = null,i_load = null,i_main = null;
   private ImageItem ii_logo,ii_load,ii_main;
   private Display display;
   private List l_lighton;
   private String[] light = {"Device 1","Device 2","Device 3","Device
4","Device 5","Device 6",
                        "Device 7","Device 8"};
   private String tmp;
   private String lightrecive;
   private BTServicAndDeviceAgent agent;
   private static final String UUID_STRING = "1122334455";
   private ChoiceGroup devices = new ChoiceGroup(null,
Choice.EXCLUSIVE);
   private boolean block = true;


      public Mobile() {
            try {
                  i_logo = Image.createImage("/giox.png");
                  i_load = Image.createImage("/load.png");
                  i_main = Image.createImage("/main.png");
                  ii_logo = new
ImageItem("",i_logo,ImageItem.LAYOUT_CENTER,"Image can't Display");
                  ii_load = new
ImageItem("",i_load,ImageItem.LAYOUT_CENTER,"Image can't Display");
                  ii_main = new
ImageItem("",i_main,ImageItem.LAYOUT_CENTER,"Image can't Display");
                  display = Display.getDisplay(this);
                  l_lighton = new List("Select
light",List.MULTIPLE,light,null);

                  //**add component area
                  f_start.append(ii_logo);
                  f_start.addCommand(c_start);
                  f_start.addCommand(c_connect);
```

```
        f_start.addCommand(c_exit);
        f_main.append(ii_main);
        f_main.addCommand(c_status);
        f_main.addCommand(c_on);
        f_main.addCommand(c_exit);
        f_status.addCommand(c_back);
        f_status.addCommand(c_ok);
        f_search.append(ii_load);
        l_lighton.addCommand(c_back);
        l_lighton.addCommand(c_ok);
        f_loaddevice.addCommand(c_select);
        f_status.setCommandListener(this);
        l_lighton.setCommandListener(this);
        f_start.setCommandListener(this);
        f_main.setCommandListener(this);
        f_loaddevice.setCommandListener(this);
                    agent = new
BTServicAndDeviceAgent(UUID_STRING,"L2CAPchat",this);
              } catch(Exception e) { System.out.println("Error :: " +
e.getMessage()); }
        }

        public void ReciveData(String message) { // override from
interface
              lightrecive = message;
              setLight(lightrecive);
              block = false;
        }

        public void FoundDevice(String[] devicesname) { // override
from interface
              int numberofdevice = devicesname.length;
              for(int i=0;i<numberofdevice;i++)
              {
                    devices.append(devicesname[i],null);
              }
              f_loaddevice.append(devices);
              display.setCurrent(f_loaddevice);
        }

    public void startApp() {
        display.setCurrent(f_start);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }

    public void commandAction(Command cmd, Displayable source) {
        tmp = "";
        if(cmd == c_start) {
              agent.StartServer();
              display.setCurrent(f_main);
        } else if(cmd == c_connect) { // connect to server (client)
              display.setCurrent(f_search);
              agent.SearchDevice();
        } else if(cmd == c_select && source == f_loaddevice) {
              int index = devices.getSelectedIndex();
              agent.SearchService(index);
```

```
            display.setCurrent(f_main);
        } else if(cmd == c_status) {
            agent.ToSendData("rxxxxxxx");
            while(block);
            display.setCurrent(f_status);
        } else if(cmd == c_on) {
        display.setCurrent(l_lighton);
        } else if(cmd == c_back) {
        display.setCurrent(f_main);
        } else if(cmd == c_ok && source == l_lighton) {
        tmp = "";
        boolean[] selected = new boolean[light.length];
        ((List)source).getSelectedFlags(selected);
        for(int i=0;i<light.length;i++) {
            if(selected[i]) {
                tmp += "1";
            } else { tmp += "0"; }
        }
        agent.ToSendData(tmp);
        System.out.println("Device :: " + tmp);
        display.setCurrent(f_main);
        } else if(cmd == c_exit) {
        agent.StopClient();
                agent.StopServer();
                agent.ReleaseResource();
          destroyApp(false);
          notifyDestroyed();
        }
    }
    public void setLight(String s) {
        display.setCurrent(l_lighton);
        boolean[] selected = new boolean[light.length];
        for(int i=0;i<light.length;i++) {
            if(s.charAt(i) == '1') {
                selected[i] = true;
            } else {
                selected[i] = false;
            }
        }
        ((List)l_lighton).setSelectedFlags(selected);
        display.setCurrent(l_lighton);
    }

} // end of class Mobile
```

```
import javax.bluetooth.*;
import java.util.*;
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;

public final class BTServicAndDeviceDiscovery implements
DiscoveryListener {
      private Vector remoteDevices = new Vector();
      private ServiceRecord serviceRecord;
      private BTServicAndDeviceAgent callback;
      private String SERVICE_NAME;
      public BTServicAndDeviceDiscovery(String
SERVICE_NAME,BTServicAndDeviceAgent callback) {
            this.SERVICE_NAME = SERVICE_NAME;
            this.callback = callback;
      }
      public void servicesDiscovered(int transID, ServiceRecord[]
servRecord) {
            for(int i = 0; i < servRecord.length; i++) {
            DataElement serviceNameElement =
servRecord[i].getAttributeValue(0x0100);//get the Service Name
            String serviceName =
(String)serviceNameElement.getValue();
            if(serviceName.equals(SERVICE_NAME)){
                serviceRecord = servRecord[0];
            }
        }
      }
    public void serviceSearchCompleted(int transID, int respCode) {
            String message = null;
        if (respCode ==
DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE) {
            message = "Device not reachable";
        }
        else if (respCode ==
DiscoveryListener.SERVICE_SEARCH_NO_RECORDS) {
            message = "Service not available";
        }
        else if (respCode ==
DiscoveryListener.SERVICE_SEARCH_COMPLETED) {
            message = "Service search completed";
        }
        else if (respCode ==
DiscoveryListener.SERVICE_SEARCH_TERMINATED) {
            message = "Service search terminated";
        }
        else if (respCode == DiscoveryListener.SERVICE_SEARCH_ERROR)
{
            message = "Service search error";
        }
        callback.serviceSearchFinished(serviceRecord, message);
      }
      public void inquiryCompleted(int discType) {
            String message = null;
        if (discType == INQUIRY_COMPLETED) {
            message = "Inquiry completed";
        } else if (discType == INQUIRY_TERMINATED) {
            message = "Inquiry terminated";
        } else if (discType == INQUIRY_ERROR) {
```

```
            message = "Inquiry error";
        }
        RemoteDevice[] devices = new
RemoteDevice[remoteDevices.size()];
        for(int i = 0; i < remoteDevices.size(); i++) {
            devices[i] = (RemoteDevice)remoteDevices.elementAt(i);
        }
            callback.deviceInquiryFinished(devices, message);
      }

    public void deviceDiscovered(RemoteDevice btDevice, DeviceClass
cod) {
            remoteDevices.addElement(btDevice);
      }
}
```

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import javax.bluetooth.*;
import java.io.*;

public final class BTServicAndDeviceAgent
{
      private String UUID_STRING;
      private String SERVICE_NAME;
      private LocalDevice localDevice;
    private DiscoveryAgent agent;
    private L2CAPConnection conn;
    private RemoteDevice[] remoteDevices;
      private boolean running = false;
      private BTCallback callback;
      private String msg;
      private BTServicAndDeviceDiscovery discoverer;
      private String[] devices;
      private BTServicAndDeviceAgent(){
      }
      public BTServicAndDeviceAgent(String UUID_STRING,String
SERVICE_NAME,BTCallback callback){
            this.UUID_STRING = UUID_STRING;
            this.SERVICE_NAME = SERVICE_NAME;
            this.callback = callback;
            discoverer = new
BTServicAndDeviceDiscovery(SERVICE_NAME,this);
            try{
            localDevice = LocalDevice.getLocalDevice();
            agent = localDevice.getDiscoveryAgent();
         }catch(BluetoothStateException bse){
                  bse.printStackTrace();
         }
      }
      public void StartServer() {
            new Thread() {
                  public void run() {
                        try{

      localDevice.setDiscoverable(DiscoveryAgent.GIAC);
                              L2CAPConnectionNotifier notifier =
(L2CAPConnectionNotifier)Connector.open("btl2cap://localhost:" +
UUID_STRING + ";name=L2CAPchat");
                              ServiceRecord record =
localDevice.getRecord(notifier);
                              String connURL =
record.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT,
false);
                              conn = notifier.acceptAndOpen();
                              new Thread(){
                                    public void run(){
                                          running = true;
                                          ToReciveData();
                                    }
                              }.start();
                        } catch(IOException ioe) {
                              ioe.printStackTrace();
                        }
                  }
            }.start();
```

```
    }
    public void StopServer() {
        running = false;
    }
    public void SearchDevice() {
            try {
            //non-blocking
            agent.startInquiry(DiscoveryAgent.GIAC, discoverer);

        } catch(BluetoothStateException bse){
           bse.printStackTrace();
        }
    }
    public void SearchService(int index) {
        int[] attrSet = {0x0100};//return service name attribute
        UUID[] uuidSet = new UUID[1];
        uuidSet[0] = new UUID(UUID_STRING, false);
        try{
            agent.searchServices(attrSet, uuidSet,
remoteDevices[index], discoverer);
        } catch(BluetoothStateException bse){
            bse.printStackTrace();
      }
    }
    public void StopClient() {
        running = false;
    }
    public void deviceInquiryFinished(RemoteDevice[] remoteDevices,
String message){
        this.remoteDevices = remoteDevices;
        devices = new String[remoteDevices.length];
      for(int i = 0; i < remoteDevices.length; i++) {
        try {
            String name =
remoteDevices[i].getFriendlyName(false);
            devices[i] = name;
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
      }
        callback.FoundDevice(devices);
    }
    public void serviceSearchFinished(ServiceRecord serviceRecord,
String message){
        String url =
serviceRecord.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT
, false);
      try{
        conn = (L2CAPConnection)Connector.open(url);
            new Thread(){
          public void run(){
                    running = true;
             ToReciveData();
          }
            }.start();
        }catch(IOException ioe) {
            ioe.printStackTrace();
        }
    }
    public void ToReciveData() {
        while(running){
```

```
            try{
                if(conn.ready()){
                    int receiveMTU = conn.getReceiveMTU();
                    byte[] data = new byte[receiveMTU];
                    int length = conn.receive(data);
                    String message = new String(data, 0, length);
                    callback.ReciveData(message);
                }
            }catch(IOException ioe){
                ioe.printStackTrace();
            }
        }
    }
    public void ToSendData(String message) {
            msg = null;
            msg = message;
            new Thread(){
                public void run(){
                    try {
            byte[] data = msg.getBytes();
            int transmitMTU = conn.getTransmitMTU();
            if(data.length <= transmitMTU){
            conn.send(data);
            }else {
                    System.out.println("Message too long!");
                        }
            } catch(IOException ioe) {
                    ioe.printStackTrace();
                    }
            }
    }.start();
    }
    public void ReleaseResource() {
            try{
                    if(localDevice != null) localDevice = null;
                    if(agent != null) agent = null;
                    if(remoteDevices != null) remoteDevices = null;
            if(conn != null) conn.close();
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
}
}
```

Source Code

ของโปรแกรมบน MCS-51

```
#include <reg51.h>
#include <stdio.h>
#define DELAY_SERIAL_SEND 100

/*------------------------------------------------
P0 Bit Registers
------------------------------------------------*/
sbit P00 = 0x80;
sbit P01 = 0x81;
sbit P02 = 0x82;
sbit P03 = 0x83;
sbit P04 = 0x84;
sbit P05 = 0x85;
sbit P06 = 0x86;
sbit P07 = 0x87;

/*------------------------------------------------
P1 Bit Registers
------------------------------------------------*/
sbit P10 = 0x90;
sbit P11 = 0x91;
sbit P12 = 0x92;
sbit P13 = 0x93;
sbit P14 = 0x94;
sbit P15 = 0x95;
sbit P16 = 0x96;
sbit P17 = 0x97;

/*------------------------------------------------
P2 Bit Registers
------------------------------------------------*/
sbit P20 = 0xA0;
sbit P21 = 0xA1;
sbit P22 = 0xA2;
sbit P23 = 0xA3;
sbit P24 = 0xA4;
sbit P25 = 0xA5;
sbit P26 = 0xA6;
sbit P27 = 0xA7;

/*------------------------------------------------
P3 Bit Registers (Mnemonics & Ports)
------------------------------------------------*/
sbit P30 = 0xB0;
sbit P31 = 0xB1;
sbit P32 = 0xB2;
sbit P33 = 0xB3;
sbit P34 = 0xB4;
sbit P35 = 0xB5;
sbit P36 = 0xB6;
sbit P37 = 0xB7;

/*------------------------------------------------
Global Variable
------------------------------------------------*/
      int c;
      int a;
      int b;
      int buf;
      int buf2;
```

```
void delay(unsigned char count)
{
      unsigned char i,j;
      for (i = 0 ; i < count ; i++)
      for (j = 0 ; j < 200 ; j++);
}

void Display_LED(unsigned char led)
{
      switch(led)
      {
            case '1':
                  P20 = 1;
            break;
            case '2':
                  P21 = 1;
            break;
            case '3':
                  P22 = 1;
            break;
            case '4':
                  P23 = 1;
            break;
            case '5':
                  P24 = 1;
            break;
            case '6':
                  P25 = 1;
            break;
            case '7':
                  P26 = 1;
            break;
            case '8':
                  P27 = 1;
            break;
            case 'a':
                  P20 = 0;
            break;
            case 'b':
                  P21 = 0;
            break;
            case 'c':
                  P22 = 0;
            break;
            case 'd':
                  P23 = 0;
            break;
            case 'e':
                  P24 = 0;
            break;
            case 'f':
                  P25 = 0;
            break;
            case 'g':
                  P26 = 0;
            break;
            case 'h':
                  P27 = 0;
            break;
            case 'r':
                  SBUF=buf;
```

```
            break;
      }
}

//////////////////////        Serial          //////////////////////
void init_serial() {
      SCON = 0x50;            // mode 1: 8-bit UART, enable receiver
      TMOD = 0x21;            //    set timer
      TH1 = 0xFD;             //    set baud rate
      TL1 = 0xFD;             //    set baud rate
      TR1 = 1;                //    set timer1 enable
      EA = 1;
      ES = 1;
}
void serial_send( int c ) {
      SBUF = c;
}
serial () interrupt 4 using 2  {
/* use registerbank 2 for interrupt */
      char c;
//////////  receive
      if (RI)  {                    /* if receiver interrupt */
            buf2 = SBUF;
            c = buf2;               /* read character */
            delay( DELAY_SERIAL_SEND );
            Display_LED(c);
            buf = P2;
            RI = 0;                 /* clear interrupt request flag */
      }
//////////  transmit
      if (TI)  {                    /* if transmitter interrupt */
      TI = 0;                       /* clear interrupt request flag */
      }
}

//////////////////////        Serial      //////////////////////////
void init() {
      P0 = 0xff;
      P1 = 0xff;
      P2 = 0xff;
      P3 = 0xff;
      init_serial();
      P2 = 0x00;
}
void main() {
      init();
      while (1) {
      }
}
```

Features

- Compatible with MCS-51™ Products
- 4K Bytes of In-System Reprogrammable Flash Memory
  - Endurance: 1,000 Write/Erase Cycles
- Fully Static Operation: 0 Hz to 24 MHz
- Three-Level Program Memory Lock
- 128 x 8-Bit Internal RAM
- 32 Programmable I/O Lines
- Two 16-Bit Timer/Counters
- Six Interrupt Sources
- Programmable Serial Channel
- Low Power Idle and Power Down Modes

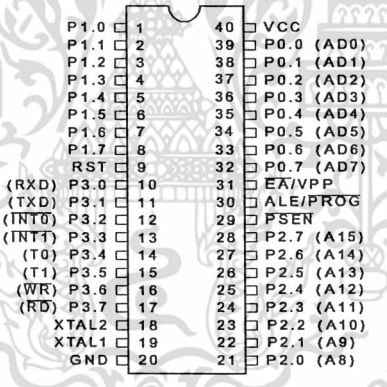
Description

The AT89C51 is a low-power, high-performance CMOS 8-bit microcomputer with 4K bytes of Flash Programmable and Erasable Read Only Memory (PEROM). The device is manufactured using Atmel's high density nonvolatile memory technology and is compatible with the industry standard MCS-51™ instruction set and pinout. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89C51 is a powerful microcomputer which provides a highly flexible and cost effective solution to many embedded control applications.

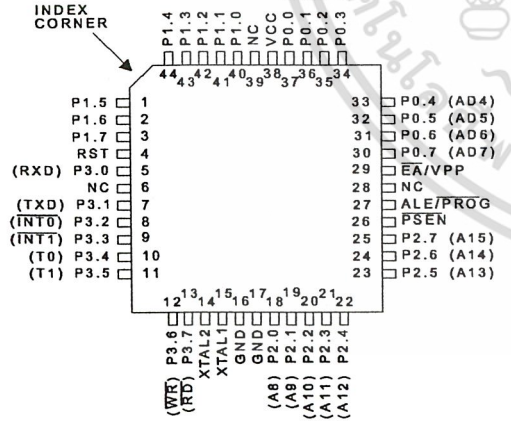
(continued)

Pin Configurations

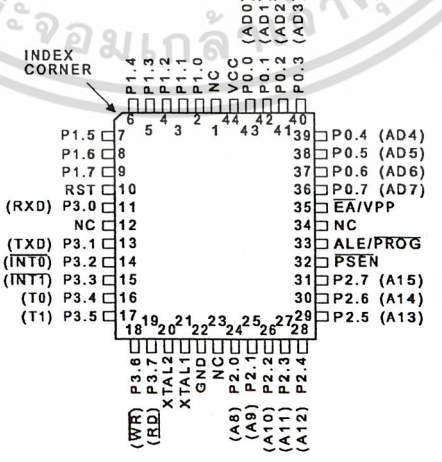
PDIP



PQFP/TQFP



PLCC

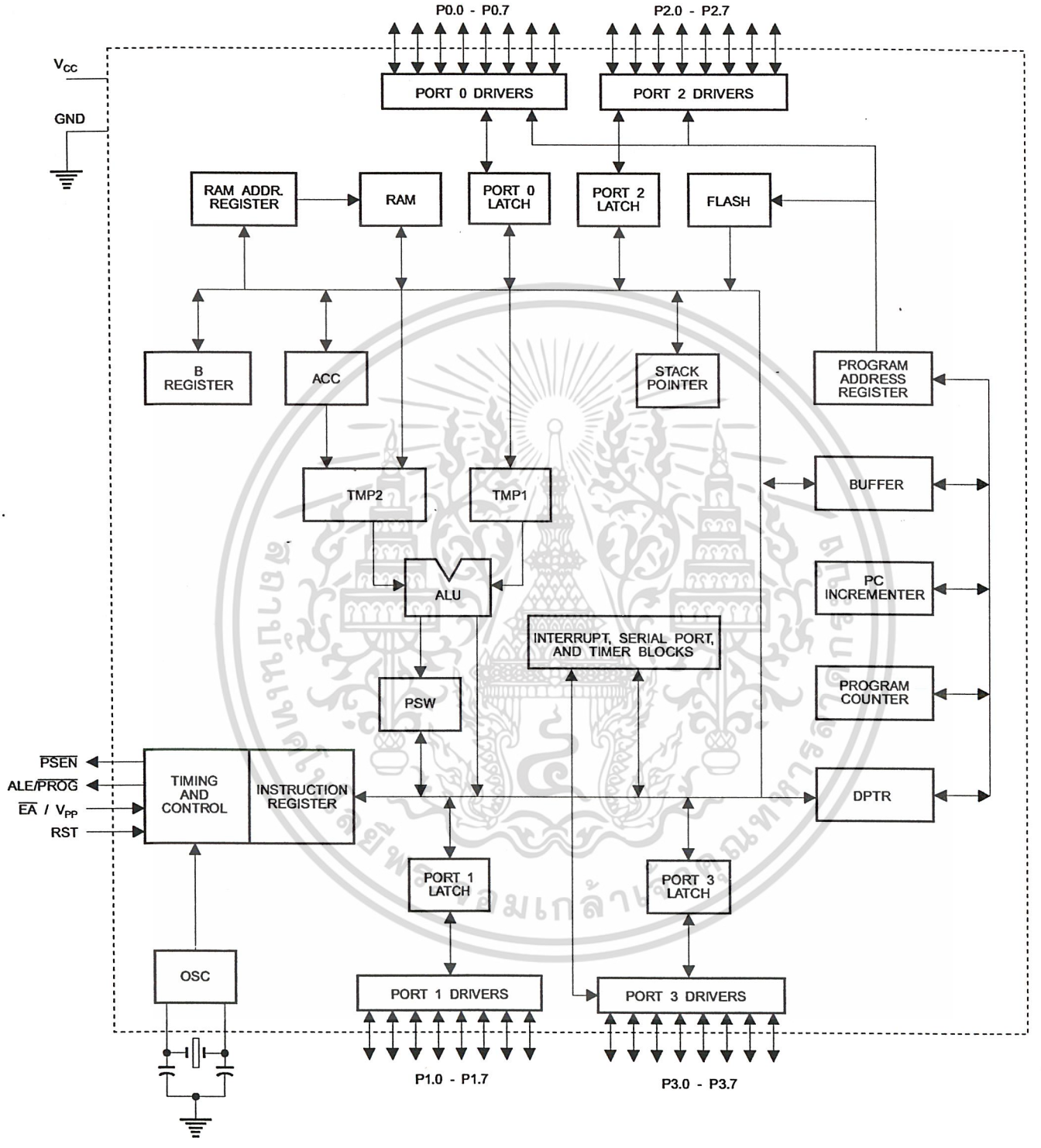


8-Bit Microcontroller with 4K Bytes Flash

AT89C51

0265F-A-12/97

ock Diagram



The AT89C51 provides the following standard features: 4K bytes of Flash, 128 bytes of RAM, 32 I/O lines, two 16-bit timer/counters, a five vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator and clock circuitry. In addition, the AT89C51 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port and interrupt system to continue functioning. The Power Down Mode saves the RAM contents but freezes the oscillator disabling all other chip functions until the next hardware reset.

Pin Description

V_{CC}
Supply voltage.

GND
Ground.

Port 0
Port 0 is an 8-bit open drain bidirectional I/O port. As an output port each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high-impedance inputs.

Port 0 may also be configured to be the multiplexed low-order address/data bus during accesses to external program and data memory. In this mode P0 has internal pullups.

Port 0 also receives the code bytes during Flash programming, and outputs the code bytes during program verification. External pullups are required during program verification.

Port 1
Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Port 1 also receives the low-order address bytes during Flash programming and verification.

Port 2
Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source four TTL inputs. When 1s are written to Port 2 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @ DPTR). In this application it uses strong internal pullups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @ RI), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits and some control signals during Flash programming and verification.

Port 3
Port 3 is an 8-bit bidirectional I/O port with internal pullups. The Port 3 output buffers can sink/source four TTL inputs. When 1s are written to Port 3 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (I_{IL}) because of the pullups.

Port 3 also serves the functions of various special features of the AT89C51 as listed below:

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

Port 3 also receives some control signals for Flash programming and verification.

RST
Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/$\overline{PROG}$
Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input ($\overline{PROG}$) during Flash programming.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory.

If desired, ALE operation can be disabled by setting bit 0 of SFR location 8EH. With the bit set, ALE is active only during a MOVX or MOVC instruction. Otherwise, the pin is weakly pulled high. Setting the ALE-disable bit has no effect if the microcontroller is in external execution mode.

$\overline{PSEN}$
Program Store Enable is the read strobe to external program memory.

When the AT89C51 is executing code from external program memory, $\overline{PSEN}$ is activated twice each machine cycle, except that two $\overline{PSEN}$ activations are skipped during each access to external data memory.

$\overline{EA}$/V_{PP}
External Access Enable. $\overline{EA}$ must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, $\overline{EA}$ will be internally latched on reset.

$\overline{EA}$ should be strapped to V_{CC} for internal program executions.

This pin also receives the 12-volt programming enable voltage (V_{PP}) during Flash programming, for parts that require 12-volt V_{PP}.

XTAL1
Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

XTAL2
Output from the inverting oscillator amplifier.

Oscillator Characteristics

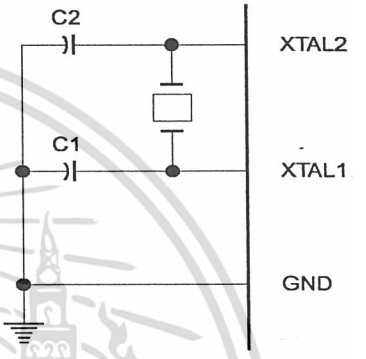
XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 1. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external clock source, XTAL2 should be left unconnected while XTAL1 is driven as shown in Figure 2. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum voltage high and low time specifications must be observed.

Idle Mode

In idle mode, the CPU puts itself to sleep while all the on-chip peripherals remain active. The mode is invoked by software. The content of the on-chip RAM and all the special functions registers remain unchanged during this mode. The idle mode can be terminated by any enabled interrupt or by a hardware reset.

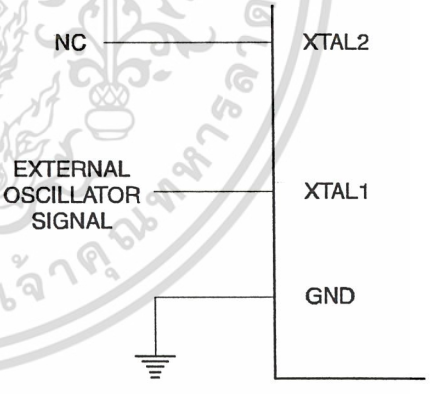
It should be noted that when idle is terminated by a hard ware reset, the device normally resumes program execution, from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write to a port pin when Idle is terminated by reset, the instruction following the one that invokes Idle should not be one that writes to a port pin or to external memory.

Figure 1. Oscillator Connections



Note: C1, C2 = 30 pF ± 10 pF for Crystals
= 40 pF ± 10 pF for Ceramic Resonators

Figure 2. External Clock Drive Configuration



Status of External Pins During Idle and Power Down Modes

Mode	Program Memory	ALE	$\overline{PSEN}$	PORT0	PORT1	PORT2	PORT3
Idle	Internal	1	1	Data	Data	Data	Data
Idle	External	1	1	Float	Data	Address	Data
Power Down	Internal	0	0	Data	Data	Data	Data
Power Down	External	0	0	Float	Data	Data	Data

Power Down Mode

In the power down mode the oscillator is stopped, and the instruction that invokes power down is the last instruction executed. The on-chip RAM and Special Function Registers retain their values until the power down mode is terminated. The only exit from power down is a hardware reset. Reset redefines the SFRs but does not change the on-chip RAM. The reset should not be activated before V_{CC} is restored to its normal operating level and must be held active long enough to allow the oscillator to restart and stabilize.

Program Memory Lock Bits

On the chip are three lock bits which can be left unprogrammed (U) or can be programmed (P) to obtain the additional features listed in the table below:

When lock bit 1 is programmed, the logic level at the $\overline{EA}$ pin is sampled and latched during reset. If the device is powered up without a reset, the latch initializes to a random value, and holds that value until reset is activated. It is necessary that the latched value of $\overline{EA}$ be in agreement with the current logic level at that pin in order for the device to function properly.

Lock Bit Protection Modes

Program Lock Bits				Protection Type
	LB1	LB2	LB3	
1	U	U	U	No program lock features.
2	P	U	U	MOVC instructions executed from external program memory are disabled from fetching code bytes from internal memory, $\overline{EA}$ is sampled and latched on reset, and further programming of the Flash is disabled.
3	P	P	U	Same as mode 2, also verify is disabled.
4	P	P	P	Same as mode 3, also external execution is disabled.

Programming the Flash

The AT89C51 is normally shipped with the on-chip Flash memory array in the erased state (that is, contents = FFH) and ready to be programmed. The programming interface accepts either a high-voltage (12-volt) or a low-voltage (V_{CC}) program enable signal. The low voltage programming mode provides a convenient way to program the AT89C51 inside the user's system, while the high-voltage programming mode is compatible with conventional third party Flash or EPROM programmers.

The AT89C51 is shipped with either the high-voltage or low-voltage programming mode enabled. The respective top-side marking and device signature codes are listed in the following table.

	V_{PP} = 12V	V_{PP} = 5V
Top-Side Mark	AT89C51 xxxx yyww	AT89C51 xxxx-5 yyww
Signature	(030H)=1EH (031H)=51H (032H)=FFH	(030H)=1EH (031H)=51H (032H)=05H

The AT89C51 code memory array is programmed byte-by-byte in either programming mode. *To program any non-blank byte in the on-chip Flash Memory, the entire memory must be erased using the Chip Erase Mode.*

Programming Algorithm: Before programming the AT89C51, the address, data and control signals should be set up according to the Flash programming mode table and Figures 3 and 4. To program the AT89C51, take the following steps.

1. Input the desired memory location on the address lines.
2. Input the appropriate data byte on the data lines.
3. Activate the correct combination of control signals.
4. Raise $\overline{EA}/V_{PP}$ to 12V for the high-voltage programming mode.
5. Pulse ALE/$\overline{PROG}$ once to program a byte in the Flash array or the lock bits. The byte-write cycle is self-timed and typically takes no more than 1.5 ms. Repeat steps 1 through 5, changing the address and data for the entire array or until the end of the object file is reached.

$\overline{Data}$ Polling: The AT89C51 features $\overline{Data}$ Polling to indicate the end of a write cycle. During a write cycle, an attempted read of the last byte written will result in the complement of the written datum on PO.7. Once the write cycle has been completed, true data are valid on all outputs, and the next cycle may begin. $\overline{Data}$ Polling may begin any time after a write cycle has been initiated.

Ready/$\overline{Busy}$: The progress of byte programming can also be monitored by the RDY/$\overline{BSY}$ output signal. P3.4 is pulled low after ALE goes high during programming to indicate BUSY. P3.4 is pulled high again when programming is done to indicate READY.

Program Verify: If lock bits LB1 and LB2 have not been programmed, the programmed code data can be read back via the address and data lines for verification. The lock bits cannot be verified directly. Verification of the lock bits is achieved by observing that their features are enabled.

Chip Erase: The entire Flash array is erased electrically by using the proper combination of control signals and by holding ALE/$\overline{PROG}$ low for 10 ms. The code array is written with all "1"s. The chip erase operation must be executed before the code memory can be re-programmed.

Reading the Signature Bytes: The signature bytes are read by the same procedure as a normal verification of locations 030H,

031H, and 032H, except that P3.6 and P3.7 must be pulled to a logic low. The values returned are as follows.

(030H) = 1EH indicates manufactured by Atmel
(031H) = 51H indicates 89C51
(032H) = FFH indicates 12V programming
(032H) = 05H indicates 5V programming

Programming Interface

Every code byte in the Flash array can be written and the entire array can be erased by using the appropriate combination of control signals. The write operation cycle is self-timed and once initiated, will automatically time itself to completion.

All major programming vendors offer worldwide support for the Atmel microcontroller series. Please contact your local programming vendor for the appropriate software revision.

Flash Programming Modes

Mode		RST	$\overline{PSEN}$	ALE/$\overline{PROG}$	$\overline{EA}/V_{PP}$	P2.6	P2.7	P3.6	P3.7
Write Code Data		H	L		H/12V	L	H	H	H
Read Code Data		H	L	H	H	L	L	H	H
Write Lock	Bit - 1	H	L		H/12V	H	H	H	H
	Bit - 2	H	L		H/12V	H	H	L	L
	Bit - 3	H	L		H/12V	H	L	H	L
Chip Erase		H	L	(1)	H/12V	H	L	L	L
Read Signature Byte		H	L	H	H	L	L	L	L

Note: 1. Chip Erase requires a 10-ms PROG pulse.

Figure 3. Programming the Flash

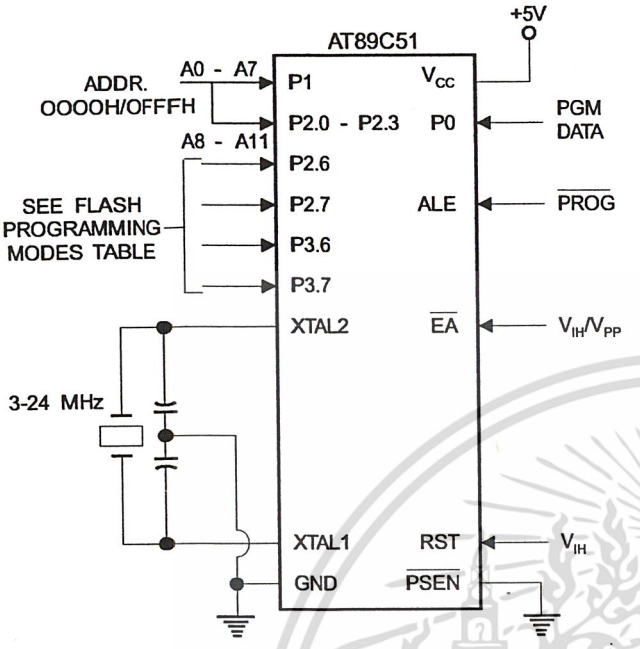
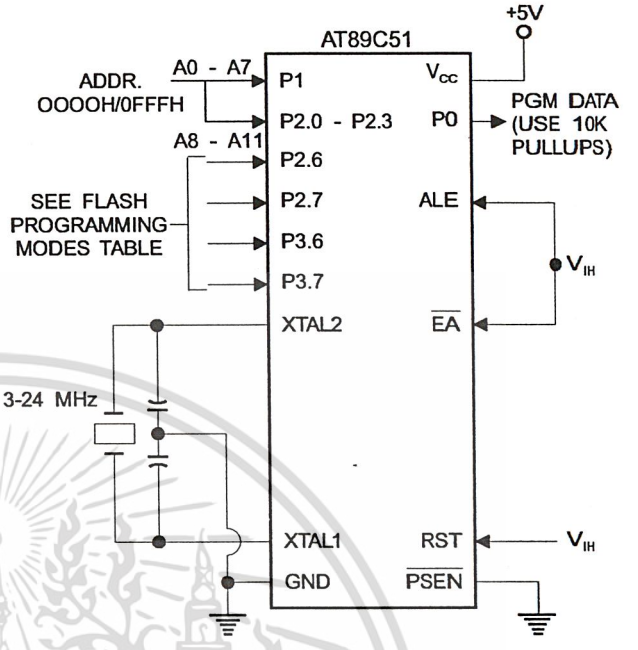


Figure 4. Verifying the Flash



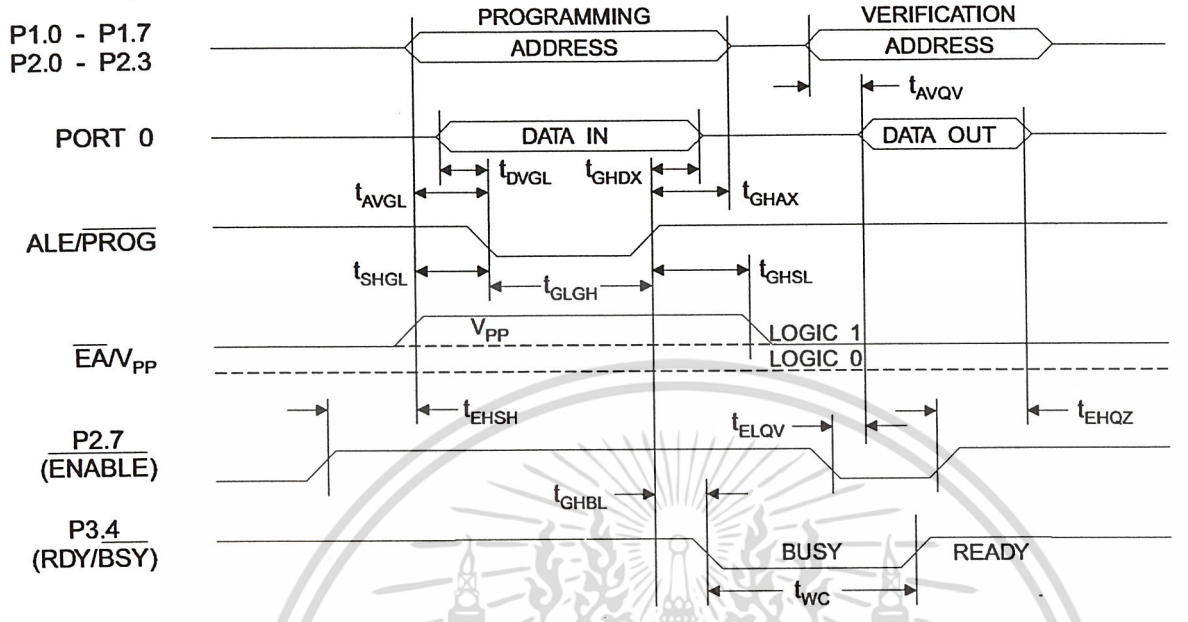
Flash Programming and Verification Characteristics

T_A = 0°C to 70°C, V_{CC} = 5.0 ± 10%

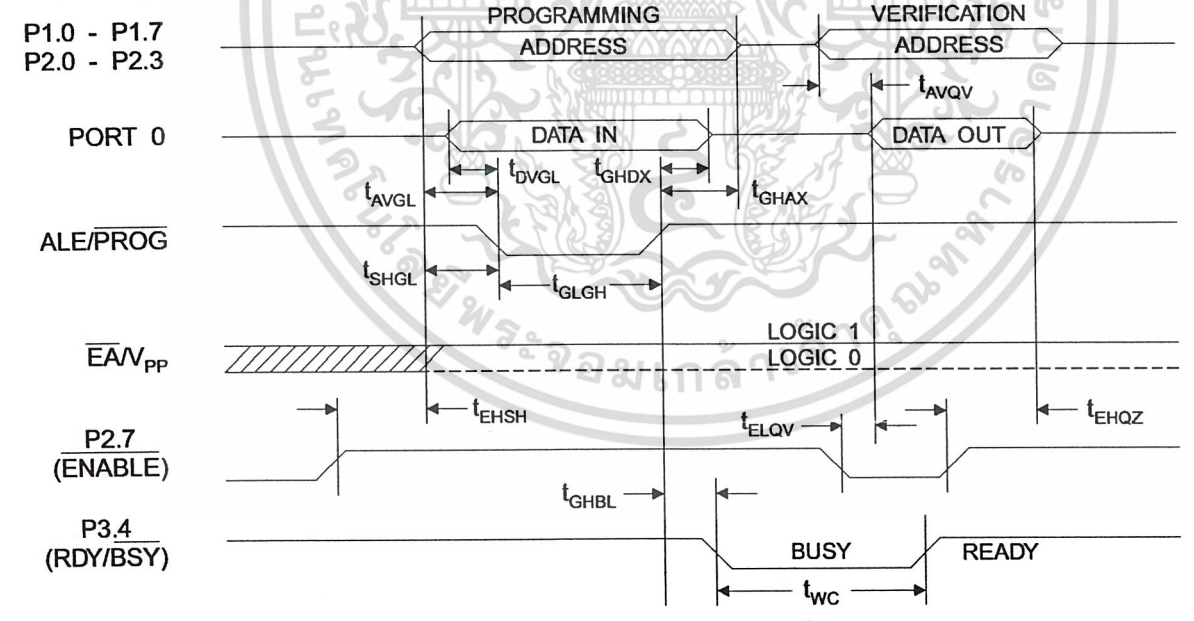
Symbol	Parameter	Min	Max	Units
V_{PP}(1)	Programming Enable Voltage	11.5	12.5	V
I_{PP}(1)	Programming Enable Current		1.0	mA
$1/t_{CLCL}$	Oscillator Frequency	3	24	MHz
t_{AVGL}	Address Setup to $\overline{PROG}$ Low	$48t_{CLCL}$		
t_{GHAX}	Address Hold After $\overline{PROG}$	$48t_{CLCL}$		
t_{DVGL}	Data Setup to $\overline{PROG}$ Low	$48t_{CLCL}$		
t_{GHDX}	Data Hold After $\overline{PROG}$	$48t_{CLCL}$		
t_{EHSH}	P2.7 ($\overline{ENABLE}$) High to V_{PP}	$48t_{CLCL}$		
t_{SHGL}	V_{PP} Setup to $\overline{PROG}$ Low	10		µs
t_{GHSL}(1)	V_{PP} Hold After $\overline{PROG}$	10		µs
t_{GLGH}	$\overline{PROG}$ Width	1	110	µs
t_{AVQV}	Address to Data Valid		$48t_{CLCL}$	
t_{ELQV}	$\overline{ENABLE}$ Low to Data Valid		$48t_{CLCL}$	
t_{EHQZ}	Data Float After $\overline{ENABLE}$	0	$48t_{CLCL}$	
t_{GHBL}	$\overline{PROG}$ High to $\overline{BUSY}$ Low		1.0	µs
t_{WC}	Byte Write Cycle Time		2.0	ms

Note: 1. Only used in 12-volt programming mode.

Flash Programming and Verification Waveforms - High Voltage Mode (V_{PP} = 12V)



Flash Programming and Verification Waveforms - Low Voltage Mode (V_{PP} = 5V)



Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin with Respect to Ground	-1.0V to +7.0V
Maximum Operating Voltage	6.6V
DC Output Current	15.0 mA

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC Characteristics

T_A = -40°C to 85°C, V_{CC} = 5.0V ± 20% (unless otherwise noted)

Symbol	Parameter	Condition	Min	Max	Units
V_{IL}	Input Low Voltage	(Except $\overline{EA}$)	-0.5	0.2 V_{CC} - 0.1	V
V_{IL1}	Input Low Voltage ($\overline{EA}$)		-0.5	0.2 V_{CC} - 0.3	V
V_{IH}	Input High Voltage	(Except XTAL1, RST)	0.2 V_{CC} + 0.9	V_{CC} + 0.5	V
V_{IH1}	Input High Voltage	(XTAL1, RST)	0.7 V_{CC}	V_{CC} + 0.5	V
V_{OL}	Output Low Voltage(1) (Ports 1,2,3)	I_{OL} = 1.6 mA		0.45	V
V_{OL1}	Output Low Voltage(1) (Port 0, ALE, $\overline{PSEN}$)	I_{OL} = 3.2 mA		0.45	V
V_{OH}	Output High Voltage (Ports 1,2,3, ALE, $\overline{PSEN}$)	I_{OH} = -60 µA, V_{CC} = 5V ± 10%	2.4		V
		I_{OH} = -25 µA	0.75 V_{CC}		V
		I_{OH} = -10 µA	0.9 V_{CC}		V
V_{OH1}	Output High Voltage (Port 0 in External Bus Mode)	I_{OH} = -800 µA, V_{CC} = 5V ± 10%	2.4		V
		I_{OH} = -300 µA	0.75 V_{CC}		V
		I_{OH} = -80 µA	0.9 V_{CC}		V
I_{IL}	Logical 0 Input Current (Ports 1,2,3)	V_{IN} = 0.45V		-50	µA
I_{TL}	Logical 1 to 0 Transition Current (Ports 1,2,3)	V_{IN} = 2V, VCC = 5V ± 10%		-650	µA
I_{LI}	Input Leakage Current (Port 0, $\overline{EA}$)	0.45 < V_{IN} < V_{CC}		±10	µA
RRST	Reset Pulldown Resistor		50	300	KΩ
C_{IO}	Pin Capacitance	Test Freq. = 1 MHz, T_A = 25°C		10	pF
I_{CC}	Power Supply Current	Active Mode, 12 MHz		20	mA
		Idle Mode, 12 MHz		5	mA
	Power Down Mode(2)	V_{CC} = 6V		100	µA
		V_{CC} = 3V		40	µA

Notes: 1. Under steady state (non-transient) conditions, I_{OL} must be externally limited as follows:
Maximum I_{OL} per port pin: 10 mA
Maximum I_{OL} per 8-bit port: Port 0: 26 mA
Ports 1, 2, 3: 15 mA
Maximum total I_{OL} for all output pins: 71 mA
If I_{OL} exceeds the test condition, V_{OL} may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test conditions.

2. Minimum V_{CC} for Power Down is 2V.

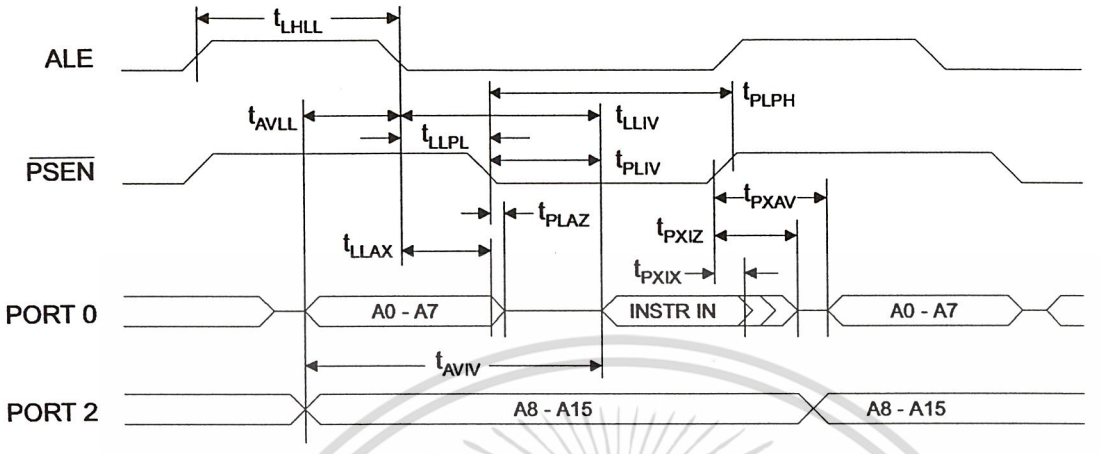
AC Characteristics

(Under Operating Conditions; Load Capacitance for Port 0, ALE/$\overline{PROG}$, and $\overline{PSEN}$ = 100 pF; Load Capacitance for all other outputs = 80 pF)

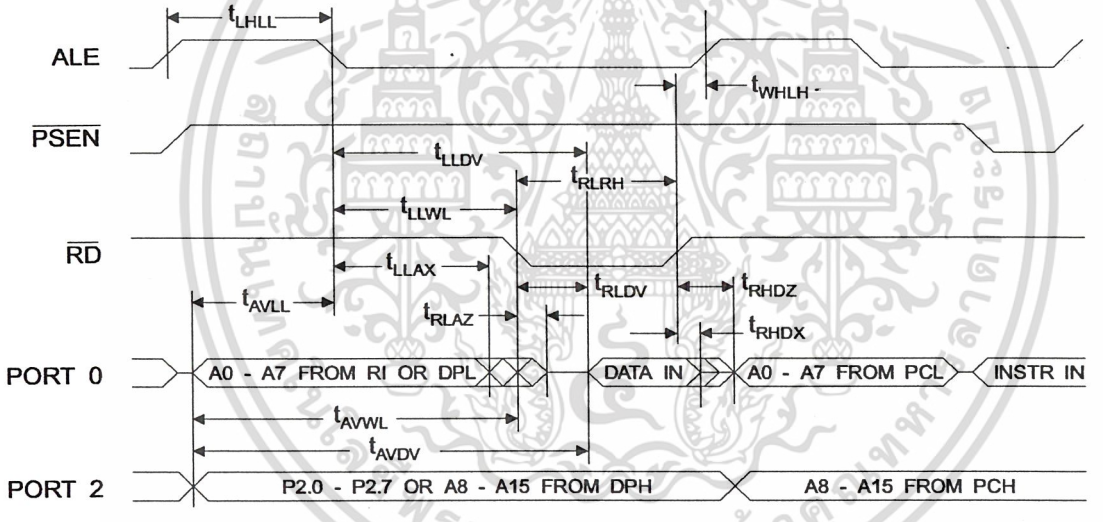
External Program and Data Memory Characteristics

Symbol	Parameter	12 MHz Oscillator		16 to 24 MHz Oscillator		Units
		Min	Max	Min	Max	
$1/t_{CLCL}$	Oscillator Frequency			0	24	MHz
t_{LHLL}	ALE Pulse Width	127		$2t_{CLCL}-40$		ns
t_{AVLL}	Address Valid to ALE Low	43		$t_{CLCL}-13$		ns
t_{LLAX}	Address Hold After ALE Low	48		$t_{CLCL}-20$		ns
t_{LLIV}	ALE Low to Valid Instruction In		233		$4t_{CLCL}-65$	ns
t_{LLPL}	ALE Low to $\overline{PSEN}$ Low	43		$t_{CLCL}-13$		ns
t_{PLPH}	$\overline{PSEN}$ Pulse Width	205		$3t_{CLCL}-20$		ns
t_{PLIV}	$\overline{PSEN}$ Low to Valid Instruction In		145		$3t_{CLCL}-45$	ns
t_{PXIX}	Input Instruction Hold After $\overline{PSEN}$	0		0		ns
t_{PXIZ}	Input Instruction Float After $\overline{PSEN}$		59		$t_{CLCL}-10$	ns
t_{PXAV}	$\overline{PSEN}$ to Address Valid	75		$t_{CLCL}-8$		ns
t_{AVIV}	Address to Valid Instruction In		312		$5t_{CLCL}-55$	ns
t_{PLAZ}	$\overline{PSEN}$ Low to Address Float		10		10	ns
t_{RLRH}	$\overline{RD}$ Pulse Width	400		$6t_{CLCL}-100$		ns
t_{WLWH}	$\overline{WR}$ Pulse Width	400		$6t_{CLCL}-100$		ns
t_{RLDV}	$\overline{RD}$ Low to Valid Data In		252		$5t_{CLCL}-90$	ns
t_{RHDX}	Data Hold After $\overline{RD}$	0		0		ns
t_{RHDZ}	Data Float After $\overline{RD}$		97		$2t_{CLCL}-28$	ns
t_{LLDV}	ALE Low to Valid Data In		517		$8t_{CLCL}-150$	ns
t_{AVDV}	Address to Valid Data In		585		$9t_{CLCL}-165$	ns
t_{LLWL}	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	200	300	$3t_{CLCL}-50$	$3t_{CLCL}+50$	ns
t_{AVWL}	Address to $\overline{RD}$ or $\overline{WR}$ Low	203		$4t_{CLCL}-75$		ns
t_{QVWX}	Data Valid to $\overline{WR}$ Transition	23		$t_{CLCL}-20$		ns
t_{QVWH}	Data Valid to $\overline{WR}$ High	433		$7t_{CLCL}-120$		ns
t_{WHQX}	Data Hold After $\overline{WR}$	33		$t_{CLCL}-20$		ns
t_{RLAZ}	$\overline{RD}$ Low to Address Float		0		0	ns
t_{WHLH}	$\overline{RD}$ or $\overline{WR}$ High to ALE High	43	123	$t_{CLCL}-20$	$t_{CLCL}+25$	ns

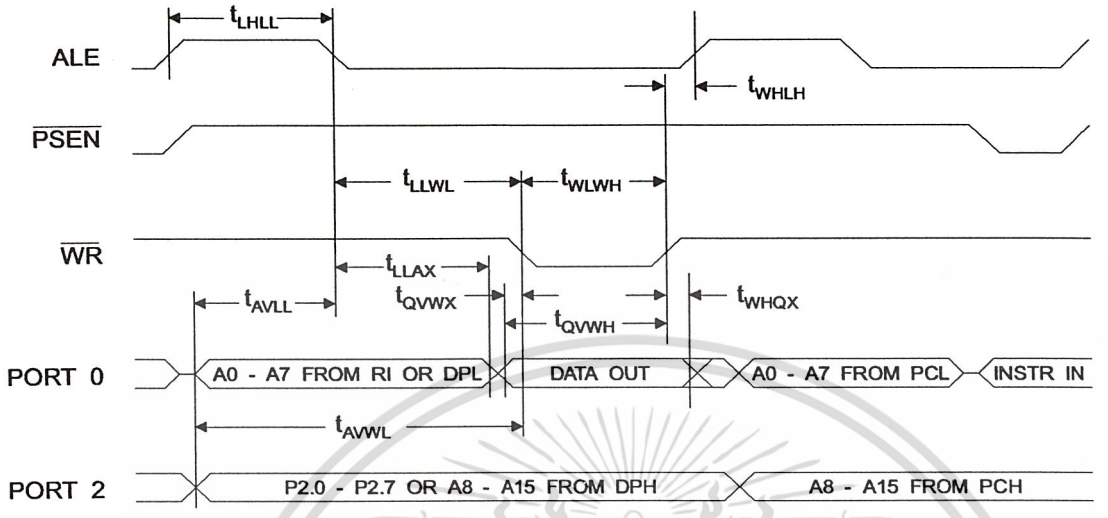
External Program Memory Read Cycle



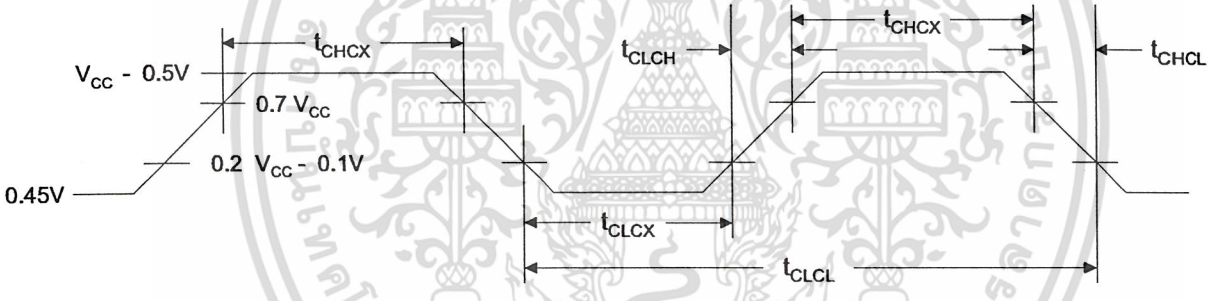
External Data Memory Read Cycle



External Data Memory Write Cycle



External Clock Drive Waveforms



External Clock Drive

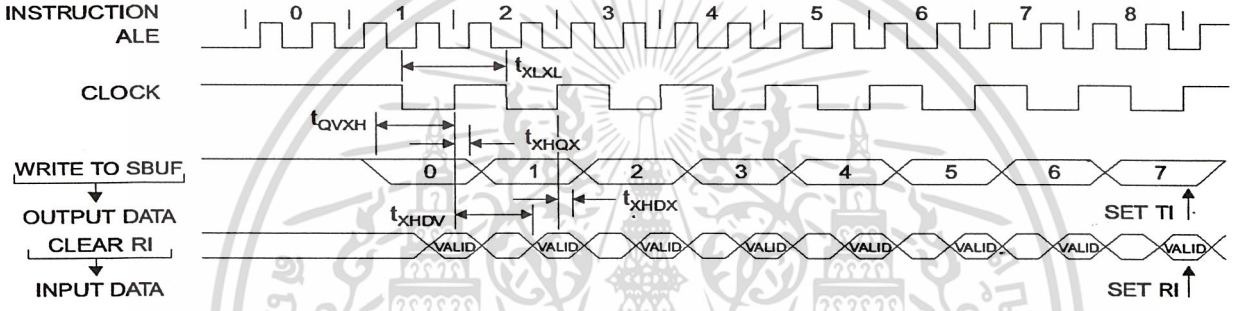
Symbol	Parameter	Min	Max	Units
$1/t_{CLCL}$	Oscillator Frequency	0	24	MHz
t_{CLCL}	Clock Period	41.6		ns
t_{CHCX}	High Time	15		ns
t_{CLCX}	Low Time	15		ns
t_{CLCH}	Rise Time		20	ns
t_{CHCL}	Fall Time		20	ns

Serial Port Timing: Shift Register Mode Test Conditions

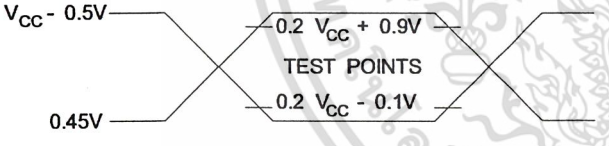
(V_{CC} = 5.0 V ± 20%; Load Capacitance = 80 pF)

Symbol	Parameter	12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
t_{XLXL}	Serial Port Clock Cycle Time	1.0		$12t_{CLCL}$		µs
t_{QVXH}	Output Data Setup to Clock Rising Edge	700		$10t_{CLCL}$-133		ns
t_{XHQX}	Output Data Hold After Clock Rising Edge	50		$2t_{CLCL}$-117		ns
t_{XHDX}	Input Data Hold After Clock Rising Edge	0		0		ns
t_{XHDV}	Clock Rising Edge to Input Data Valid		700		$10t_{CLCL}$-133	ns

Shift Register Mode Timing Waveforms

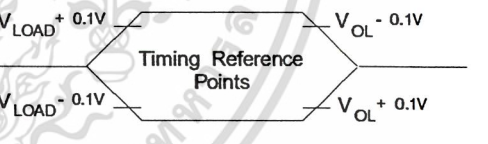


AC Testing Input/Output Waveforms[1]



Note: 1. AC Inputs during testing are driven at V_{CC} - 0.5V for a logic 1 and 0.45V for a logic 0. Timing measurements are made at V_{IH} min. for a logic 1 and V_{IL} max. for a logic 0.

Float Waveforms[1]



Note: 1. For timing purposes, a port pin is no longer floating when a 100 mV change from load voltage occurs. A port pin begins to float when 100 mV change from the loaded V_{OH}/V_{OL} level occurs.

Ordering Information

Speed (MHz)	Power Supply	Ordering Code	Package	Operation Range
12	5V ± 20%	AT89C51-12AC	44A	Commercial
		AT89C51-12JC	44J	(0°C to 70°C)
		AT89C51-12PC	40P6	
		AT89C51-12QC	44Q	
		AT89C51-12AI	44A	Industrial
		AT89C51-12JI	44J	(-40°C to 85°C)
		AT89C51-12PI	40P6	
		AT89C51-12QI	44Q	
		AT89C51-12AA	44A	Automotive
		AT89C51-12JA	44J	(-40°C to 105°C)
		AT89C51-12PA	40P6	
		AT89C51-12QA	44Q	
16	5V ± 20%	AT89C51-16AC	44A	Commercial
		AT89C51-16JC	44J	(0°C to 70°C)
		AT89C51-16PC	40P6	
		AT89C51-16QC	44Q	
		AT89C51-16AI	44A	Industrial
		AT89C51-16JI	44J	(-40°C to 85°C)
		AT89C51-16PI	40P6	
		AT89C51-16QI	44Q	
		AT89C51-16AA	44A	Automotive
		AT89C51-16JA	44J	(-40°C to 105°C)
		AT89C51-16PA	40P6	
		AT89C51-16QA	44Q	
20	5V ± 20%	AT89C51-20AC	44A	Commercial
		AT89C51-20JC	44J	(0°C to 70°C)
		AT89C51-20PC	40P6	
		AT89C51-20QC	44Q	
		AT89C51-20AI	44A	Industrial
		AT89C51-20JI	44J	(-40°C to 85°C)
		AT89C51-20PI	40P6	
		AT89C51-20QI	44Q	

Ordering Information

Speed (MHz)	Power Supply	Ordering Code	Package	Operation Range
24	5V ± 20%	AT89C51-24AC	44A	Commercial (0°C to 70°C)
		AT89C51-24JC	44J	
		AT89C51-24PC	44P6	
		AT89C51-24QC	44Q	
		AT89C51-24AI	44A	Industrial (-40°C to 85°C)
		AT89C51-24JI	44J	
		AT89C51-24PI	44P6	
		AT89C51-24QI	44Q	

Package Type	
44A	44 Lead, Thin Plastic Gull Wing Quad Flatpack (TQFP)
44J	44 Lead, Plastic J-Leaded Chip Carrier (PLCC)
40P6	40 Lead, 0.600" Wide, Plastic Dual Inline Package (PDIP)
44Q	44 Lead, Plastic Gull Wing Quad Flatpack (PQFP)

Order this document by ULN2803/D

MOTOROLA

ULN2803
ULN2804

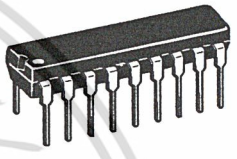
Octal High Voltage, High Current Darlington Transistor Arrays

OCTAL PERIPHERAL DRIVER ARRAYS

SEMICONDUCTOR TECHNICAL DATA

The eight NPN Darlington connected transistors in this family of arrays are ideally suited for interfacing between low logic level digital circuitry (such as TTL, CMOS or PMOS/NMOS) and the higher current/voltage requirements of lamps, relays, printer hammers or other similar loads for a broad range of computer, industrial, and consumer applications. All devices feature open–collector outputs and free wheeling clamp diodes for transient suppression.

The ULN2803 is designed to be compatible with standard TTL families while the ULN2804 is optimized for 6 to 15 volt high level CMOS or PMOS.



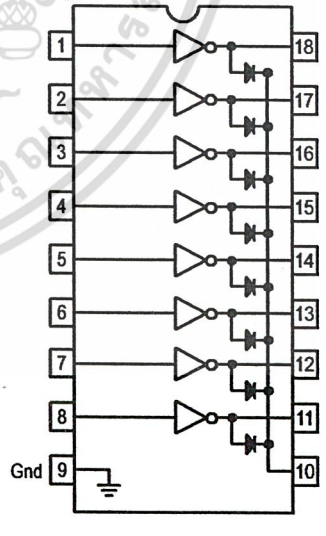
A SUFFIX
PLASTIC PACKAGE
CASE 707

MAXIMUM RATINGS (T_A = 25°C and rating apply to any one device in the package, unless otherwise noted.)

Rating	Symbol	Value	Unit
Output Voltage	V_O	50	V
Input Voltage (Except ULN2801)	V_I	30	V
Collector Current – Continuous	I_C	500	mA
Base Current – Continuous	I_B	25	mA
Operating Ambient Temperature Range	T_A	0 to +70	°C
Storage Temperature Range	T_{stg}	–55 to +150	°C
Junction Temperature	T_J	125	°C

$R_{\theta JA}$ = 55°C/W
Do not exceed maximum current limit per driver.

PIN CONNECTIONS



ORDERING INFORMATION

Device	Characteristics		
	Input Compatibility	V_{CE}(Max)/I_C(Max)	Operating Temperature Range
ULN2803A ULN2804A	TTL, 5.0 V CMOS 6 to 15 V CMOS, PMOS	50 V/500 mA	T_A = 0 to + 70°C

Rev 1

ELECTRICAL CHARACTERISTICS (T_A = 25°C, unless otherwise noted)

Characteristic		Symbol	Min	Typ	Max	Unit
Output Leakage Current (Figure 1)		I_{CEX}				µA
(V_O = 50 V, T_A = +70°C)	All Types		–	–	100	
(V_O = 50 V, T_A = +25°C)	All Types		–	–	50	
(V_O = 50 V, T_A = +70°C, V_I = 6.0 V)	ULN2802		–	–	500	
(V_O = 50 V, T_A = +70°C, V_I = 1.0 V)	ULN2804		–	–	500	
Collector–Emitter Saturation Voltage (Figure 2)		$V_{CE(sat)}$				V
(I_C = 350 mA, I_B = 500 µA)	All Types		–	1.1	1.6	
(I_C = 200 mA, I_B = 350 µA)	All Types		–	0.95	1.3	
(I_C = 100 mA, I_B = 250 µA)	All Types		–	0.85	1.1	
Input Current – On Condition (Figure 4)		$I_{I(on)}$				mA
(V_I = 17 V)	ULN2802		–	0.82	1.25	
(V_I = 3.85 V)	ULN2803		–	0.93	1.35	
(V_I = 5.0 V)	ULN2804		–	0.35	0.5	
(V_I = 12 V)	ULN2804		–	1.0	1.45	
Input Voltage – On Condition (Figure 5)		$V_{I(on)}$				V
(V_{CE} = 2.0 V, I_C = 300 mA)	ULN2802		–	–	13	
(V_{CE} = 2.0 V, I_C = 200 mA)	ULN2803		–	–	2.4	
(V_{CE} = 2.0 V, I_C = 250 mA)	ULN2803		–	–	2.7	
(V_{CE} = 2.0 V, I_C = 300 mA)	ULN2803		–	–	3.0	
(V_{CE} = 2.0 V, I_C = 125 mA)	ULN2804		–	–	5.0	
(V_{CE} = 2.0 V, I_C = 200 mA)	ULN2804		–	–	6.0	
(V_{CE} = 2.0 V, I_C = 275 mA)	ULN2804		–	–	7.0	
(V_{CE} = 2.0 V, I_C = 350 mA)	ULN2804		–	–	8.0	
Input Current – Off Condition (Figure 3) (I_C = 500 µA, T_A = +70°C)	All Types	$I_{I(off)}$	50	100	–	µA
DC Current Gain (Figure 2) (V_{CE} = 2.0 V, I_C = 350 mA)	ULN2801	h_{FE}	1000	–	–	–
Input Capacitance		C_I	–	15	25	pF
Turn–On Delay Time (50% E_I to 50% E_O)		t_{on}	–	0.25	1.0	µs
Turn–Off Delay Time (50% E_I to 50% E_O)		t_{off}	–	0.25	1.0	µs
Clamp Diode Leakage Current (Figure 6) (V_R = 50 V)	T_A = +25°C T_A = +70°C	I_R	–	–	50 100	µA
Clamp Diode Forward Voltage (Figure 7) (I_F = 350 mA)		V_F	–	1.5	2.0	V

TEST FIGURES

(See Figure Numbers in Electrical Characteristics Table)

Figure 1.

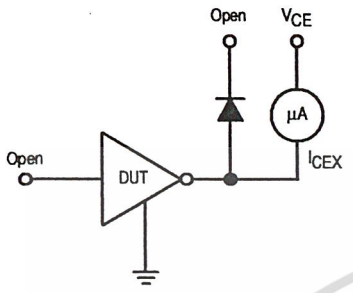


Figure 2.

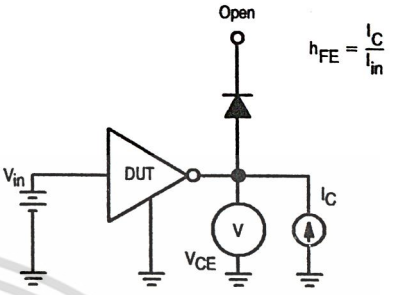


Figure 3.

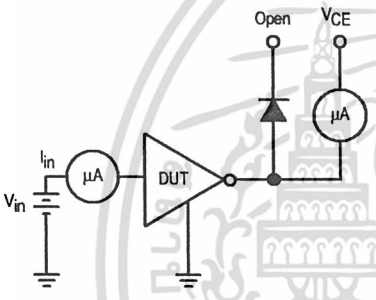


Figure 4.

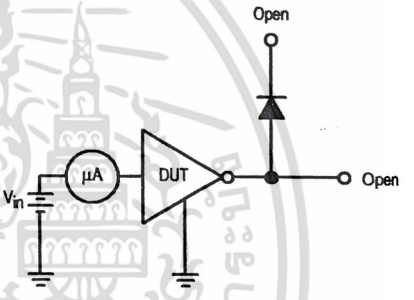


Figure 5.

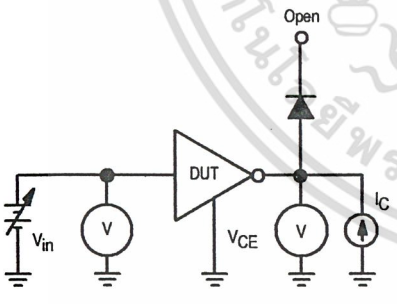


Figure 6.

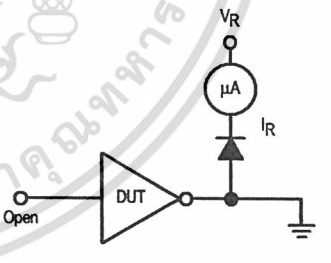
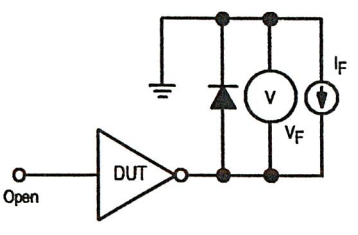


Figure 7.



TYPICAL CHARACTERISTIC CURVES – T_A = 25°C, unless otherwise noted

Output Characteristics

Figure 8. Output Current versus Saturation Voltage

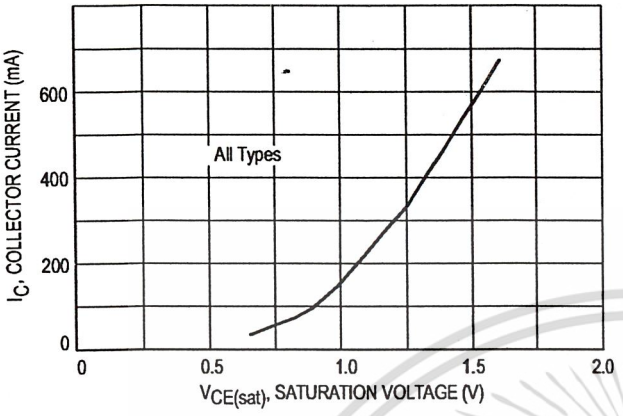
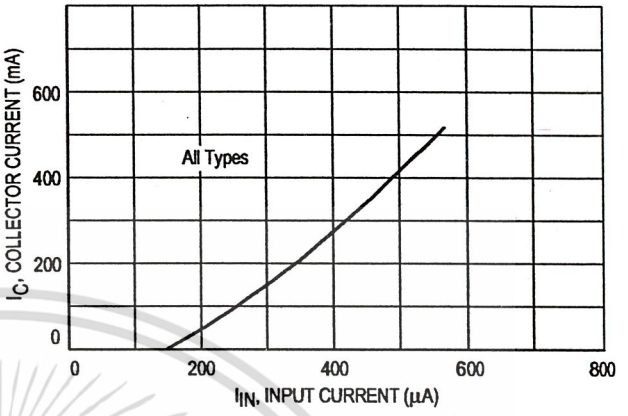


Figure 9. Output Current versus Input Current



Input Characteristics

Figure 10. ULN2803 Input Current versus Input Voltage

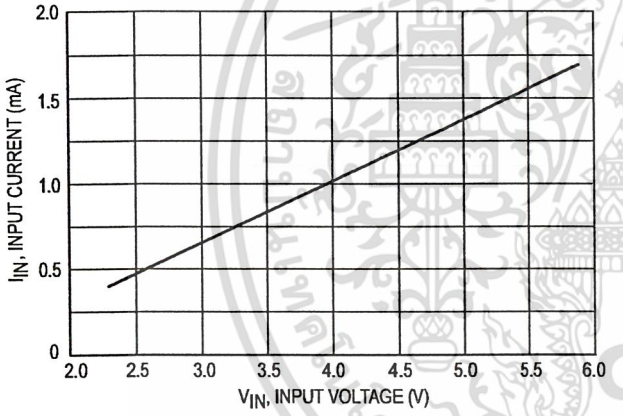


Figure 11. ULN2804 Input Current versus Input Voltage

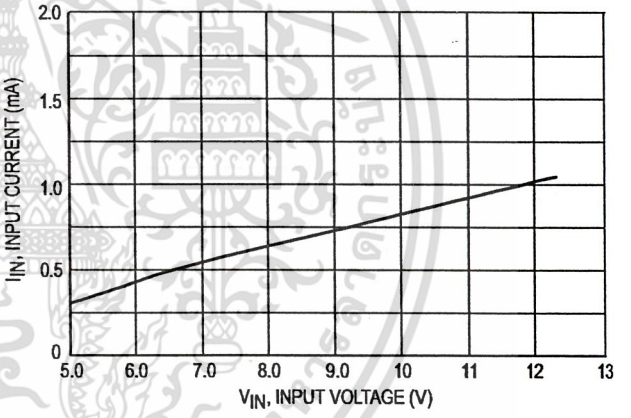
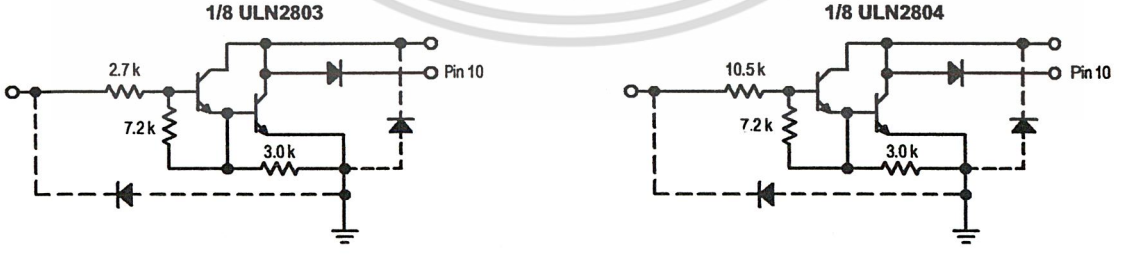
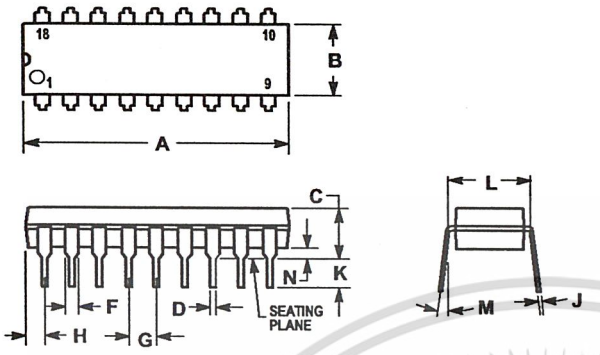


Figure 12. Representative Schematic Diagrams



OUTLINE DIMENSIONS

A SUFFIX
PLASTIC PACKAGE
CASE 707–02
ISSUE C



NOTES:
1. POSITIONAL TOLERANCE OF LEADS (D), SHALL BE WITHIN 0.25 (0.010) AT MAXIMUM MATERIAL CONDITION, IN RELATION TO SEATING PLANE AND EACH OTHER.
2. DIMENSION L TO CENTER OF LEADS WHEN FORMED PARALLEL.
3. DIMENSION B DOES NOT INCLUDE MOLD FLASH.

	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	22.22	23.24	0.875	0.915
B	6.10	6.60	0.240	0.260
C	3.56	4.57	0.140	0.180
D	0.36	0.56	0.014	0.022
F	1.27	1.78	0.050	0.070
G	2.54 BSC		0.100 BSC	
H	1.02	1.52	0.040	0.060
J	0.20	0.30	0.008	0.012
K	2.92	3.43	0.115	0.135
L	7.62 BSC		0.300 BSC	
M	0°	15°	0°	15°
N	0.51	1.02	0.020	0.040

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036. 1–800–441–2447 or 602–303–5454

MFAX: RMFAX0@email.sps.mot.com – TOUCHTONE 602–244–6609
INTERNET: http://Design–NET.com

JAPAN: Nippon Motorola Ltd.; Tatsumi–SPD–JLDC, 6F Seibu–Butsuryu–Center, 3–14–2 Tatsumi Koto–Ku, Tokyo 135, Japan. 03–81–3521–8315

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park, 51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852–26629298