การควบคุมอุปกรณ์บลูทูธขั้นพื้นฐาน

BLUETOOTH-BASED DEVICE CONTROL

โดย

นายจิรพงษ์ เปี่ยมศรี

นางสาวชฎาพร สุขภูวงศ์

นางสาวอาภาภรณ์ บุญธรรม

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

การควบคุมอุปกรณ์บลูทูธขั้นพื้นฐาน
BLUETOOTH-BASED DEVICE CONTROL

โดย

นายจิรพงษ์ เปี่ยมศรี 43010062

นางสางชฎาพร สุขภูวงศ์ 43010081

นางสาวอาภาภรณ์ บุญธรรม 43010543

อาจารย์ที่ปรึกษา

รศ.ดร.กอบชัย เดชหาญ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

ปริญญานิพนธ์ปีการศึกษา 2546

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การควบคุมอุปกรณ์บลูทูธขั้นพื้นฐาน**

BLUETOOTH-BASED DEVICE CONTROL

ผู้จัดทำ

1. **นายจิรพงศ์ เปี่ยมศรี 43010062**
2. **นางสาวชฎาพร สุขภูวงศ์ 43010081**
3. **นางสาวอาภาภรณ์ บุญธรรม 43010543**

..... อาจารย์ที่ปรึกษา

(รศ.ดร.กอบชัย เดชหาญ)

การควบคุมอุปกรณ์บลูทูธขั้นพื้นฐาน

BLUETOOTH-BASED DEVICE CONTROL

โดย นายจิรพงศ์ เปี่ยมศรี 43010062
นางสาวชฎาพร สุขภูวงศ์ 43010081
นางสาวอาภาภรณ์ บุญธรรม 43010543

อาจารย์ที่ปรึกษา รศ.ดร.กอบชัย เดชหาญ

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ เป็นการนำเสนอการศึกษาการทำงานของอุปกรณ์บลูทูธและฝึกเขียนโปรแกรมสำหรับติดต่อระหว่างเครื่องคอมพิวเตอร์กับไมโครคอนโทรลเลอร์โดยผ่านอุปกรณ์บลูทูธ ซึ่งข้อดีของการใช้อุปกรณ์บลูทูธ คือ บลูทูธเป็นอุปกรณ์ที่มีขนาดเล็ก ใช้กำลังงานไฟฟ้าต่ำ และสามารถติดตั้งได้ง่าย โครงงานนี้จึงนับว่าเป็นตัวอย่างที่ดีในการพัฒนาโปรแกรมเพื่อใช้งานกับอุปกรณ์บลูทูธ และอุปกรณ์สื่อสารแบบไร้สายประเภทอื่นๆต่อไป

ABSTRACT

This project concerns about the bluetooth-based device control and coding of the program using to practicing for interface between computer and microcontroller by bluetooth device. The advantage of Bluetooth technology is used the small device, consumes low electric power and easy to install. This project is an example for developing the program for using with bluetooth device and other wireless device.

สารบัญ

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

หน้า

สารบัญตาราง

บทที่ 1

บทนำ

1.1 ความสำคัญและที่มาของปัญหา

ความสะดวกสบายนั้นเป็นสิ่งที่มนุษย์ทั่วไปต้องการ จึงทำให้มนุษย์ได้ทำการคิดประดิษฐ์สิ่งต่างๆ เพื่อมาอำนวยความสะดวกสบายตอบสนองความต้องการของตนเอง ในช่วงแรกของการพัฒนาอุปกรณ์อิเล็กทรอนิกส์นั้น อุปกรณ์ต่างๆจำเป็นต้องมีการต่อสายต่างๆเพื่อไว้ใช้ในการควบคุมหรือแลกเปลี่ยนข้อมูลระหว่างอุปกรณ์ จึงเกิดปัญหาในเรื่องของการจัดวางสาย เป็นเหตุให้มีการคิดค้นและพัฒนาเทคโนโลยีไร้สายขึ้นมา ในปัจจุบันเทคโนโลยีไร้สายที่นิยมใช้กันอยู่กับอุปกรณ์ส่วนมากคือเทคโนโลยีไร้สายอินฟาเรด ซึ่งมีข้อจำกัดในแง่ของทิศทางการให้ทิศทางกับอุปกรณ์ ด้วยเหตุนี้จึงมีความคิดที่จะเอาเทคโนโลยีไร้สายที่มีชื่อเรียกว่า " เทคโนโลยีบลูทูธ (Bluetooth) " มาพัฒนาการสร้างเป็นอุปกรณ์ไร้สาย โดยเทคโนโลยีนี้ใช้หลักการการส่งคลื่นวิทยุที่อยู่ในย่านความถี่ระหว่าง 2,400 ถึง 2,483.5 เมกะเฮิรตซ์ ข้อดีของเทคโนโลยีนี้คือมีขนาดเล็กราคาถูกและมีความสามารถในการรับส่งสัญญาณได้ดีกว่าเทคโนโลยีไร้สายอินฟาเรด แต่ก็มีข้อจำกัดเรื่องระยะทางของการส่งสัญญาณซึ่งมีระยะประมาณ 7-10 เมตร ดังนั้นจึงเหมาะสมกับการใช้เทคโนโลยีนี้ในบริเวณบ้านหรือในตัวอาคารที่มีบริเวณไม่กว้างนัก

เนื่องจากเทคโนโลยี บลูทูธ นั้นเป็นเทคโนโลยีใหม่ โครงงานนี้จึงมุ่งที่จะทำการศึกษาเทคโนโลยีบลูทูธ โดยมุ่งเน้นไปที่การศึกษาและทำการทดลองสื่อสารระหว่างไมโครคอนโทรลเลอร์กับคอมพิวเตอร์ผ่านอุปกรณ์บลูทูธสองตัว โดยกำหนดให้ไมโครคอนโทรลเลอร์เป็นลูกข่าย (Client) แล้วส่งคำสั่งร้องขอไปยังคอมพิวเตอร์ซึ่งเป็นแม่ข่าย (Server)

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อศึกษาหลักการติดต่อสื่อสารแบบไร้สายโดยอาศัยเทคโนโลยีบลูทูธ
2. สามารถทำการเชื่อมต่อระหว่างไมโครคอนโทรลเลอร์กับคอมพิวเตอร์โดยอาศัยอุปกรณ์บลูทูธได้
3. สามารถเขียนโปรแกรมบนไมโครคอนโทรเลอร์และนำไปใช้ได้

บทที่ 2

ทฤษฎีและหลักการ

ในปัจจุบันนี้ โลกของเรากำลังก้าวเข้าสู่ยุคดิจิตอล เนื่องจากสัญญาณดิจิตอลทำให้มนุษย์เราสามารถส่งถ่ายข้อมูลชนิดต่างๆ จากที่หนึ่งไปยังอีกที่หนึ่งด้วยเวลาอันรวดเร็วเพียงพริบตาโดยยังคงสามารถคงไว้ซึ่งความถูกต้องของข้อมูลไว้ได้ การส่งถ่ายข้อมูลดิจิตอลที่กล่าวมานั้น สามารถส่งผ่านตัวกลางได้หลายชนิด ในที่นี้จะขอกล่าวถึงการส่งผ่านข้อมูลแบบไร้สาย โดยการส่งข้อมูลไร้สายที่เป็นที่นิยมอย่างแพร่หลายขณะนี้ได้แก่การใช้แสงอินฟาเรด เนื่องจากสามารถส่งผ่านข้อมูลด้วยความเร็วสูงได้ ประกอบกับความไม่ยุ่งยากของวงจรที่ใช้งาน แต่ข้อเสียของการใช้งานอินฟาเรดคือข้อจำกัดด้านระยะทางและตำแหน่งระหว่างอุปกรณ์ทั้งสอง เนื่องจากการใช้แสงอินฟาเรดเป็นตัวกลางนั้นสามารถถูกรบกวนจากแสงภายนอกได้ง่าย ทำให้ระยะทางที่ใช้งาน ไม่สามารถเพิ่มเติมได้มากนัก อีกทั้งตำแหน่งของอุปกรณ์ทั้งสองจะต้องอยู่ในแนวเส้นตรงเดียวกันและต้องไม่มีอะไรมาตัดขวางลำแสงจึงจะสามารถใช้งานได้ จากข้อเสียดังกล่าว ทำให้เกิดความคิดที่จะนำคลื่นวิทยุมาใช้ส่งข้อมูลแทน เพราะคลื่นวิทยุสามารถรับส่งข้อมูลที่มีความเร็วสูงได้ มีระยะทางในการใช้งานได้ไกล(ขึ้นอยู่กับกำลังส่ง) จำนวนของอุปกรณ์ที่สามารถเชื่อมต่อถึงกันในเวลาหนึ่งๆที่มีได้มากกว่าสองตัว และความอิสระในการวางอุปกรณ์ไว้ที่จุดใดก็ได้ที่คลื่นวิทยุสามารถแพร่กระจายไปถึง

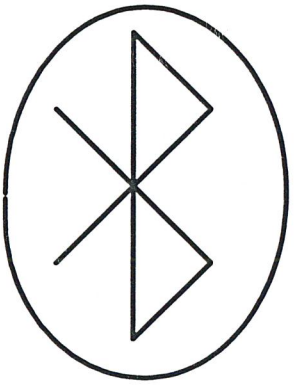
การนำคลื่นวิทยุมาใช้รับส่งข้อมูลสำหรับอุปกรณ์ระยะใกล้ ในปัจจุบันนี้มีกำหนดเป็นมาตรฐานอยู่ 3 ระบบด้วยกัน ระบบแรกคือ IEEE 802.11 หรือที่เรียกกันว่า Wireless LAN (WLAN) ออกแบบมาเพื่อสร้างระบบเน็ตเวิร์คไร้สายทดแทนการใช้สาย UTP หรือสายโคแอกเชียลที่เชื่อมต่อกับอุปกรณ์ต่างๆ เข้ากับระบบ LAN แบบอีเทอร์เนต (Ethernet) ระบบที่สองก็คือ HomeRF เป็นมาตรฐานที่กำหนดขึ้นเพื่อทดแทนการเชื่อมต่อของเครื่องใช้ไฟฟ้าที่อยู่ภายในบ้าน ซึ่งรูปแบบข้อมูลที่รับส่งกันนั้นจะมีทั้งภาพและเสียงรวมกันอยู่ ทำให้ต้องการความเร็วในการส่งข้อมูลที่สูงมาก และระบบสุดท้ายซึ่งเป็นระบบที่ใช้สำหรับโครงงานนี้ก็คือ บลูทูธ (Bluetooth) เป็นมาตรฐานที่กำหนดขึ้นเพื่อทดแทนการใช้สายเชื่อมต่อระหว่างอุปกรณ์ในระยะใกล้ อาทิ เช่น การเชื่อมต่อระหว่างพรินเตอร์กับเครื่องคอมพิวเตอร์ เครื่อง PDA กับโทรศัพท์มือถือ หรือระหว่างโทรศัพท์มือถือกับชุดหูฟัง สมอลทอร์ก เป็นต้น

เทคโนโลยี บลูทูธ เป็นเทคโนโลยีสำหรับการเชื่อมต่ออุปกรณ์แบบไร้สายที่น่าจับตามองเป็นอย่างยิ่งในปัจจุบัน ทั้งในเรื่องความสะดวกในการใช้งานสำหรับผู้ใช้ทั่วไป และประสิทธิภาพในการทำงาน เนื่องจากเทคโนโลยี บลูทูธ มีราคาถูก ใช้พลังงานน้อย และใช้เทคโนโลยี short-range ซึ่งในอนาคต จะถูกนำมาใช้ในการพัฒนาไปสู่การแทนที่อุปกรณ์ต่างๆ ที่ใช้สายเคเบิล เช่น ชุดหูฟังสมอลทอร์กสำหรับโทรศัพท์เคลื่อนที่ เป็นต้น

2.1 จุดมุ่งหมายของบลูทูธ

บลูทูธ เป็นมาตรฐานเปิด (Open Specification) ที่ออกแบบมาเพื่อใช้สำหรับการส่งข้อมูลไร้สายระยะใกล้ (Short Range Wireless) ถูกกำหนดขึ้นโดย Bluetooth SIG โดยมีจุดมุ่งหมายหลักในการวางมาตรฐานหลัก ๆ ดังนี้

- เป็นมาตรฐานเปิด ที่ทุกคนสามารถใช้ข้อมูลที่ส่งผ่านบลูทูธนี้ได้โดยไม่ต้องเสียค่าธรรมเนียมใด ๆ ทั้งสิ้น เพื่อให้เกิดความแพร่หลายในการใช้งาน และมีการพัฒนาระบบได้อย่างรวดเร็ว
- รับส่งข้อมูลไร้สายในระยะใกล้ คืออุปกรณ์ต่าง ๆ สามารถส่งข้อมูลถึงกันได้โดยไม่ต้องใช้สาย
- สามารถรองรับการรับส่งเสียงและข้อมูลได้ในเวลาเดียวกัน นั่นคือระบบจะต้องมีความเร็วในการรับส่งข้อมูลเพียงพอสำหรับการส่งเสียงและตัวข้อมูลไปพร้อม ๆ กันได้
- สามารถใช้งานได้ในทุกที่ทั่วโลก หมายความว่าอุปกรณ์ที่ผลิตตามมาตรฐานของ บลูทูธ ไม่ว่าจะผลิตจากผู้ผลิตใด หรืออยู่ ณ ตำแหน่งใดบนโลก สามารถใช้งานร่วมกันได้ จากจุดมุ่งหมายในข้อนี้ ทำให้ต้องใช้ความถี่คลื่นวิทยุที่สามารถใช้งานได้ในทุกๆประเทศ และมีการกำหนดสัญลักษณ์บลูทูธขึ้น ดังรูปที่ 2.1



รูปที่ 2.1 สัญลักษณ์ของ บลูทูธ

2.2 กำเนิดเทคโนโลยีบลูทูธ

เทคโนโลยี บลูทูธ เริ่มก่อขึ้นจากความคิดของกลุ่มวิศวกรของบริษัท อิริคสัน (Ericsson) ในปี ค.ศ. 1994 โดยมีจุดมุ่งหมายเพื่อศึกษาวิธีการรับส่งข้อมูลแบบไร้สายโดยใช้คลื่นวิทยุที่มีราคาไม่แพงและใช้พลังงานน้อย เพื่อทดแทนการใช้สายเชื่อมระหว่างโทรศัพท์มือถือกับอุปกรณ์เสริมต่างๆ ต่อมาในปีค.ศ. 1998 ก็มีการตั้งกลุ่ม Bluetooth SIG (Bluetooth Special Interest Group) ขึ้นเพื่อวางแนวทางของมาตรฐาน บลูทูธ ทั้งหมด โดยมีบริษัทยักษ์ใหญ่ 5 บริษัทเป็นสมาชิกยุคก่อตั้งได้แก่

- Ericsson Mobile Communication AB.

- Intel Corp.
- IBM Corp.
- Toshiba Corp.
- Nokia Mobile Phones.

และในเดือน กรกฎาคม ปี 1999 กลุ่มบริษัทเหล่านี้ได้ทำการเผยแพร่ Bluetooth Specification Version 1.0 นอกจากนี้ก็มีการเปิดรับสมาชิกของกลุ่มเพิ่มขึ้นเรื่อยๆ จนปัจจุบันนี้มีสมาชิกอยู่มากกว่า 1800 บริษัท Bluetooth SIG ได้ทำการแบ่งกลุ่มพัฒนาออกเป็น 3 กลุ่ม โดยกลุ่มที่ 1 ทำการแก้ไข บลูทูธเวอร์ชัน 1.0 , กลุ่มที่ 2 ทำการพัฒนาเพิ่มเติมในเวอร์ชัน 1.0 และ กลุ่มที่ 3 ทำการพัฒนา บลูทูธเวอร์ชัน 2.0 โดยในเวอร์ชัน 2.0 ของ บลูทูธ นั้น ถูกคาดหวังให้มีอัตราการส่งข้อมูลสูงขึ้น (ระหว่าง 2 ถึง 10 Mb/s) รวมไปถึงการส่งข้อมูลที่เป็นมัลติมีเดีย (multimedia) ซึ่งจะเป็นที่แพร่หลายในอนาคต

ในส่วนของชื่อ บลูทูธ (Bluetooth) นั้นได้มาจากชื่อของ Harald Blatand กษัตริย์ของเดนมาร์ก ซึ่งในช่วงที่ครองราชย์อยู่นั้นสามารถรวมเดนมาร์กและนอร์เวย์เข้าด้วยกัน เปรียบเสมือนเทคโนโลยี บลูทูธ ที่จะรวมอุตสาหกรรมด้านการสื่อสารและคอมพิวเตอร์เข้าด้วยกัน จากชื่อ Blatand เมื่อพิจารณาการออกเสียงจากภาษาดั้งเดิมโดยผิดเพี้ยนน้อยที่สุดก็จะได้ชื่อ Bluetooth และสาเหตุที่เลือกใช้ชื่อนี้ก็เพราะว่า อิริคสัน ซึ่งเป็นผู้เริ่มเทคโนโลยีมี บริษัทแม่อยู่ที่เดนมาร์กนั่นเอง

2.3 การสร้างโครงข่ายเชื่อมต่อ

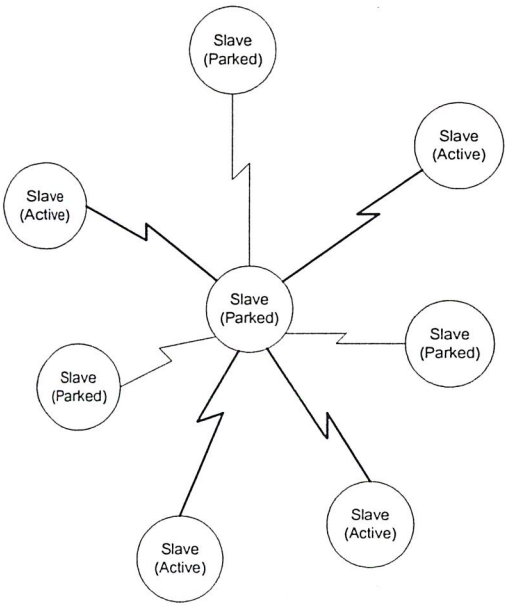
การเชื่อมต่อระหว่างอุปกรณ์ตามมาตรฐาน บลูทูธ จะอยู่ในรูปแบบของ มาสเตอร์ – สเลฟ (Master – Slave) โดยเมื่อเชื่อมต่อกันสำเร็จจะเปรียบเสมือนการสร้างเน็ตเวิร์คเสมือนขนาดเล็กขึ้น ซึ่งมีชื่อเรียกว่า พิโคเน็ท (Piconet) ตามข้อกำหนดของ บลูทูธ กำหนดให้ในแต่ละพิโคเน็ทมีอุปกรณ์ที่เป็นมาสเตอร์ได้เพียงตัวเดียวแต่สามารถเชื่อมต่อกับอุปกรณ์ที่เป็นสเลฟที่กำลังทำงานอยู่ได้ 7 ตัว และเชื่อมต่อกับสเลฟที่พักการทำงานหรืออยู่ในโหมดปาร์ค (Park) ได้ถึง 256 ตัว

สาเหตุที่มีมาสเตอร์ได้เพียงตัวเดียวก็เพราะว่า อุปกรณ์ที่ทำหน้าที่เป็นมาสเตอร์จะเป็นตัวควบคุมจังหวะการรับส่งข้อมูลของอุปกรณ์ทุก ๆ ตัวในพิโคเน็ท โดยสเลฟจะส่งข้อมูลกลับมายังมาสเตอร์ได้หลังจากสิ้นสุดการส่งข้อมูลจากมาสเตอร์เท่านั้น คือถ้ามาสเตอร์ไม่ส่งข้อมูลไปหาสเลฟ สเลฟก็จะไม่มีสิทธิ์ตอบกลับ ซึ่งสเลฟจะไม่สามารถเชื่อมต่อถึงกันได้โดยตรง แต่จะสามารถติดต่อถึงกันได้ผ่านทางมาสเตอร์เท่านั้น และคุณสมบัติแบบมาสเตอร์ – สเลฟนี้จะปรากฏอยู่ในการสื่อสารระดับล่างเท่านั้น ในระดับที่สูงขึ้นไปจนถึงตัวโปรแกรมที่ติดต่อกับผู้ใช้โดยตรงจะมองไม่เห็นความสัมพันธ์ส่วนนี้

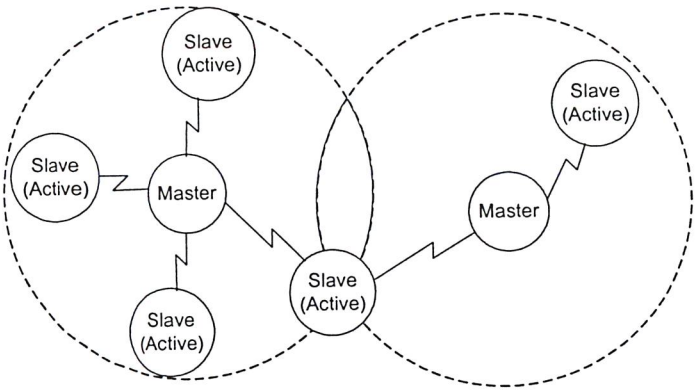
และเนื่องจากเน็ตเวิร์คเสมือนระหว่างอุปกรณ์ของบลูทูธ นี้สร้างขึ้นในอากาศรอบ ๆ อุปกรณ์ที่เป็นมาสเตอร์ในลักษณะทรงกลม ดังนั้นจึงมีความเป็นไปได้ที่จะเกิดการซ้อนทับกันของพิโคเน็ทสองวง ซึ่งถ้าเกิดความต้องการที่จะเชื่อมต่อพิโคเน็ท 2 วงนี้เข้าด้วยกันจะเรียกเน็ตเวิร์คที่สร้างขึ้นใหม่นี้ว่า สแกทเทอร์เน็ท

(Scatternet) ส่งผลให้มีรัศมีการส่งสัญญาณเพิ่มขึ้นเนื่องจากสามารถส่งสัญญาณจากวงพิโคเน็ทหนึ่งไปยังอีกวงหนึ่งได้ โดยการเชื่อมต่อระหว่างพิโคเน็ททั้งสองจะเกิดขึ้นได้ 2 ทางคือ มาสเตอร์ของพิโคเน็ทหนึ่งไปเป็นสเลฟของอีกพิโคเน็ทหนึ่ง และสเลฟของพิโคเน็ทหนึ่งไปเป็นสเลฟของอีกพิโคเน็ทหนึ่งพร้อม ๆ กัน ดังแสดงในรูปที่ 2.2 เป็นตัวอย่างการเชื่อมต่อระหว่างมาสเตอร์กับสเลฟ และ รูปที่ 2.3 เป็นตัวอย่างของเครือข่ายสแกทเทอร์เน็ท

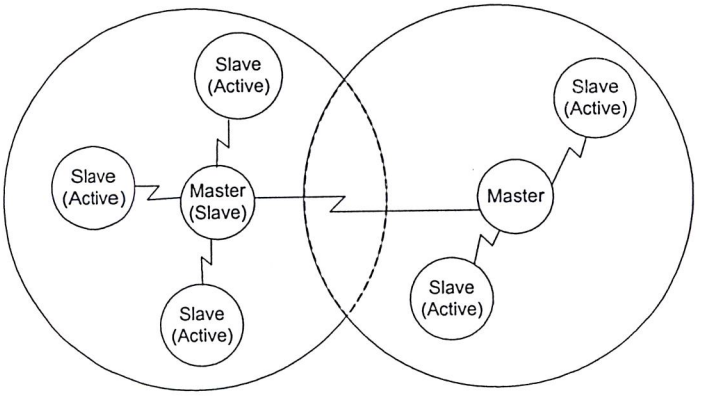
สแกทเทอร์เน็ทจะมีจำนวนวงของพิโคเน็ทได้มากที่สุดเพียง 10 วง เท่านั้น ดังนั้น ขอบเขตสูงสุดของการติดต่อจากอุปกรณ์ บลูทูธ ตัวหนึ่งไปยังอีกตัวหนึ่ง คือ ประมาณ 100 เมตร เนื่องจากใน 1 วงพิโคเน็ท มีรัศมีประมาณ 10 เมตร



รูปที่ 2.2 โครงข่ายพิโคเน็ทและการเชื่อมต่อระหว่ามาสเตอร์ - สเลฟ



(ก)



(ข)

รูปที่ 2.3 (ก) อุปกรณ์ตัวหนึ่งเป็นสเลฟในทั้งสองพิโคเน็ท
(ข) สแกทเทอร์เน็ทที่มาสเตอร์ของพิโคเน็ทหนึ่ง
เป็นสเลฟในอีกพิโคเน็ทหนึ่ง

2.4 โหมดการทำงานและการประหยัดพลังงาน

การทำงานที่เป็นสเลฟจะแบ่งออกเป็น 4 โหมดเพื่อลดการใช้งานพลังงานของอุปกรณ์ โดยมีโหมดต่างๆ ได้แก่ แอกตีฟ (Active) , สนิฟ (Sniff) , โฮลด์ (Hold) และปาร์ก (Park) แต่ละโหมดจะมีการตอบสนองต่ออุปกรณ์ที่เป็นมาสเตอร์แตกต่างๆกันดังนี้

(1) *แอกตีฟโหมด* อุปกรณ์ที่อยู่ในโหมดนี้จะคอยรับข้อมูลทุกๆ อย่างจากมาสเตอร์ เพื่อซิงโครไนซ์จังหวะการตอบกลับของตัวเองเข้ากับตัวมาสเตอร์ ซึ่งจะส่งข้อมูลตอบกลับทันทีหลังจากสิ้นสุดการส่งข้อมูลจากมาสเตอร์ การทำงนในโหมดนี้จะมีการตอบสนองที่เร็วที่สุดแต่ก็ใช้พลังงานมากที่สุด เพราะว่าอุปกรณ์ต้องรอรับข้อมูลตลอดเวลา และต้องส่งข้อมูลตอบกลับทุกๆ แพ็กเกจ

(2) *สนิฟโหมด* โหมดนี้เป็นการพยายามลดการใช้พลังงานของอุปกรณ์ลง โดยตัวอุปกรณ์จะปลี่ยนเป็นโหมดแอกตีฟเป็นช่วงๆ กล่าวคือ มาสเตอร์จะยอมที่จะส่งข้อมูลมายังสเลฟในโหมดนี้มาเป็นคาบเวลา หมายความว่า สเลฟในโหมดนี้จะคอยรับข้อมูลเป็นช่วงต้นของคาบเวลาที่กำหนดเท่านั้น ถ้าได้รับข้อมูลมาก็จะตอบกลับ แต่ถ้าไม่ได้รับก็จะสามารถหลับ (Sleep) รอไปจนถึงคาบเวลาถัดไป ความเร็วในการตอบสนองของโหมดนี้จะขึ้นอยู่กับคาบเวลาที่กำหนด ถ้าคาบเวลาสั้นก็จะตอบสนองเร็วแต่ก็จะประหยัดพลังงานได้น้อย

(3) *โฮลด์โหมด* ในโหมดนี้ สเลฟจะหยุดรอรับข้อมูลทุกอย่างจากมาสเตอร์ เป็นเวลาเท่ากับช่วงเวลาโฮลด์ (Hold Time) ที่กำหนด ซึ่งในช่วงเวลาที่โฮลด์อยู่นี้อุปกรณ์สามารถซิงโครไนซ์กัลป์ปบมาสเตอร์ตัวอื่นก่อนเพื่อทำงานบางอย่างแล้วค่อยกลับมาซิงโครไนซ์กับมาสเตอร์ตัวเดิมให้ทันก็ได้ ความเร็วของการตอบสนองขึ้นอยู่กับช่วงเวลาโฮลด์

(4) *ปาร์กโหมด* โหมดนี้เป็นโหมดที่ประหยัดพลังงานมากที่สุด แต่เวลาในการตอบสนองช้าที่สุด เนื่องจากอุปกรณ์จะอยู่ในสถานะพักการทำงาน ไม่สามารถตอบกลับไปยังมาสเตอร์ได้ ต้องรอการเปลี่ยนโหมดจากมาสเตอร์ก่อนเท่านั้นจึงจะสามารถส่งข้อมูลกลับได้ แต่การทำงานนี้ทำให้มาสเตอร์ 1 ตัวสามารถเชื่อมต่อกับอุปกรณ์อื่นๆ ได้มากกว่า 7 ตัว โดยการเปลี่ยนโหมดไปมา (ต้องการติดต่อกับอุปกรณ์ตัวใดก็ค่อย เปลี่ยนโหมดมาทำงาน)

รูปที่ 2.4 ความสัมพันธ์ระหว่างความเร็วในการตอบสนอง และการใช้พลังงาน ในโหมดการทำงานทั้ง 4 โหมด

นอกจากการควบคุมจังหวะการรับส่งของอุปกรณ์เพื่อลดการใช้พลังงาน บลูทูธ ยังออกแบบการควบคุมพลังงานของอุปกรณ์ทั้งสองฝั่ง คือแทนที่ฝั่งส่งจะส่งสัญญาณด้วยความแรงสูงเสมอ ทางฝั่งรับก็จะตรวจสอบความแรงของสัญญาณที่รับได้เข้ามา และส่งข้อมูลกลับไปยังฝั่งส่งเพื่อให้ตัวส่งลดความแรงในการส่งลง ให้เหลือขนาดที่พอเหมาะไม่มากเกินไป ทำให้สามารถประหยัดพลังงานได้มากขึ้น

2.5 ช่วงความถี่ที่ใช้งาน

หลักการพื้นฐานสำหรับการส่งข้อมูลผ่านคลื่นวิทยุก็คือ การมอดูเลท (Modulate) หรือถ้าพูดอย่างง่ายๆ ก็คือ การแฝงเอาข้อมูลรวมเข้าไปกับคลื่นวิทยุแล้วส่งออกอากาศไป เมื่อไปถึงปลายทาง ตัวรับก็แยกตัวข้อมูลออกมาจากคลื่นวิทยุแล้วนำข้อมูลที่ได้ไปใช้งาน คลื่นวิทยุที่ทำหน้าที่เป็นตัวพาข้อมูลไปยังปลายทางนี้เราเรียกว่าคลื่นพาหะ ส่วนกระบวนการในการมอดูเลทข้อมูลเข้าไปในคลื่นพาหะนั้นก็มีอยู่หลายวิธี ซึ่งแต่ละวิธีก็จะมีข้อดีข้อเสียและความเหมาะสมกับงานแต่ละอย่างแตกต่างกันไป

จากเป้าหมายหลักข้อหนึ่งของ บลูทูธ ที่จะทำให้สามารถใช้งานได้ในทุก ๆ ประเทศ ความถี่ที่จะใช้เป็นคลื่นพาหะจึงจำเป็นต้องอยู่ในช่วงที่ทั่วโลกเปิดให้ใช้งานได้โดยไม่ต้องขออนุญาต ช่วงความถี่ดังกล่าวคือ ช่วงความถี่ ISM (Industrial, Scientific and Medical Band : ISM Band) ซึ่งมีความถี่ใช้งานอยู่ในช่วง 2.4 กิกะเฮิรตซ์ (GHz) เนื่องจากช่วงความถี่นี้เป็นช่วงความถี่ที่สงวนไว้ใช้งานทางด้านอุตสาหกรรมวิทยาศาสตร์และการแพทย์ ที่ทุกประเทศทั่วโลกเปิดให้ใช้งานได้

ช่วงความถี่ ISM นั้นถูกกำหนดขอบเขตไว้ในช่วง 2,400 ถึง 2,4835 เมกะเฮิรตซ์ โดยมี Lower Guard Band (LGB) เทากับ 2 เมกะเฮิรตซ์ และ Upper Guard Band (UGB) เท่ากับ 3.5 เมกะเฮิรตซ์ และจะถูกแบ่งเป็นช่อง ๆ เพื่อให้อุปกรณ์หลาย ๆ ตัวทำงานพร้อมกันได้ตามหลักของการทำ Frequency Division Multiplex (FDM) โดยแต่ละช่องจะมีความกว้างหรือแบนวิดธ์ เท่ากับ 1 เมกะเฮิรตซ์

จากข้อกำหนดดังกล่าวจะได้ความถี่ใช้งานเริ่มต้นที่ 2,402 เมกะเฮิรตซ์ (2,400 + LGB) และความถี่ใช้งานสุดท้ายอยู่ที่ 2,480 เมกะเฮิรตซ์ (2,483.5 - UGB) เมื่อให้ความกว้างของแต่ละช่องสัญญาณเท่ากับ 1 เมกะเฮิรตซ์ จะได้จำนวนช่องสัญญาณ ทั้งหมดเท่ากับ 79 ช่อง แต่เนื่องจากยังมีบางประเทศ เช่น ฝรั่งเศส ที่ไม่เปิดช่วงความถี่ ISM เต็มช่วง (โดยเหลือช่องสัญญาณให้ใช้งานได้เพียง 23 ช่อง) แต่คาดว่าจะมีการอนุญาตให้ใช้งานได้เต็มช่วงในอนาคต ดังแสดงดังตารางที่ 2.1

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 2.1 ความถี่และช่องสัญญาณ ISM Band

นอกจากข้อกำหนดด้านความถี่ที่จะทำให้อุปกรณ์ต่าง ๆ สามารถใช้งานได้ทั่วโลกแล้ว ยังมีข้อจำกัดอีกส่วนหนึ่งที่แต่ละประเทศมีการตั้งข้อกำหนดไว้ นั่นคือเรื่องของกำลังส่งของอุปกรณ์ ซึ่งจะส่งผลโดยตรงต่อระยะห่างไกลสุดที่สามารถเชื่อมต่อกันได้อย่างที่เข้าใจกันได้ง่าย ๆ ว่าถ้ากำลังส่งแรงระยะห่างสูงสุดระหว่าง

อุปกรณ์ที่ส่งได้ก็จะมากตาม แต่เนื่องจากบลูทูธ ออกแบบมาสำหรับการส่งข้อมูลในระยะใกล้ ทำให้อุปกรณ์ไม่ต้องใช้กำลังส่งที่สูงมากซึ่งประเทศต่างๆ ให้ใช้งานได้โดยไม่ต้องขออนุญาต (คลาส 3)

ตารางที่ 2.2 แสดงให้เห็นการจัดแบ่งอุปกรณ์ออกเป็นคลาสต่าง ๆ ตามกำลังส่งสูงสุดของอุปกรณ์ ซึ่งจะเห็นได้ว่าระยะห่างไกลสุดที่สามารถทำงานได้สูงถึง 100 เมตรสำหรับอุปกรณ์คลาส 1 แต่ในปัจจุบันอุปกรณ์ที่ผลิตออกมาส่วนใหญ่จะอยู่ในคลาส 3 เพราะมีกำลังส่งที่ไม่สูงมากซึ่งหมายถึงการใช้พลังงานที่ไม่มาก ทำให้สามารถนำอุปกรณ์เหล่านี้ไปใช้งานร่วมกับอุปกรณ์พกพาที่มีความต้องการอุปกรณ์ที่ใช้พลังงานน้อยเพื่อประหยัดพลังงานแบตเตอรี่

คลาส	กำลังส่งสูงสุด	กำลังส่งต่ำสุด	ระยะการใช้งาน(โดยประมาณ)
1	100 mW (20dBm)	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 2.2 การแบ่งคลาสตามอุปกรณ์ของกำลังส่ง

2.6 เทคนิคการส่งข้อมูล

เมื่อได้ข้อตกลงด้านความถี่ที่ใช้งานจำนวนช่องสัญญาณ และแบนด์วิดธ์ของแต่ละช่องสัญญาณมาเป็นกรอบแล้ว ต่อไปก็คือการเลือกวิธีมอดูเลทข้อมูลเข้าไปกับคลื่นพาหะ เพราะการมอดูเลทแต่ละแบบจะส่งผลต่อความเร็วในการส่งข้อมูลด้วย และเนื่องจากความจำกัดในด้านแบนด์วิดธ์ของข้อมูลที่กว้างเพียง 1 เมกะเฮิรตซ์ต่อช่องสัญญาณ บวกกับความต้องการความเร็วในการส่งข้องมูลที่สูงที่สุด Bluetooth จึงได้เลือกใช้การมอดูเลทแบบ Gaussian Frequency – Shift Keying (GFSK)

การมอดูเลทด้วยวิธีนี้สามารถส่งข้อมูลได้ 1 บิตต่อความถี่คลื่นพาหะ 1 เฮิรตซ์ นั่นหมายความว่าแต่ละช่องสัญญาณสามารถส่งข้อมูลได้ที่ความเร็ว 1 เมกะบิตต่อวินาที (Mbit/s) โดยถ้าบิตข้อมูลเป็น ‘1’ จะเกิดการเปลี่ยนแปลงความถี่ทางลบจากความถี่พาหะ

การรับส่งข้อมูลจะแบ่งข้อมูลออกเป็นแพ็กเกจย่อย ๆ แล้วส่งในแบบฮาร์ฟดูเพล็กซ์ (Half Duplex) เพื่อประหยัดช่องสัญญาณ (มิฉะนั้นต้องใช้สองช่องสัญญาณเพื่อส่งและรับข้อมูลได้พร้อม ๆ กัน) จังหวะการรับส่งข้อมูลทั้งหมดกำหนดโดยอุปกรณ์ที่เป็นมาสเตอร์ในลักษณะของการโพล ซึ่งอุปกรณ์ที่เป็นสเลฟจะต้องตอบกลับมายังมาสเตอร์ในทุก ๆ แพ็กเกจเพื่อให้มาสเตอร์รู้ว่ายังคงสามารถติดต่อกับสเลฟอยู่ได้

เมื่อมีการแบ่งข้อมูลออกเป็นแพ็กเกจ ทำให้แต่ละแพ็กเกจต้องมีข้อมูลส่วนหัว (Header) เพิ่มเข้ามาเพื่อให้ทางฝั่งรับสามารถประกอบข้อมูลทั้งหมดเข้าด้วยกันได้อย่างถูกต้อง นอกจากนั้นก่อนการส่งแต่ละครั้งจะต้องมีการส่งข้อมูลเพื่อทำการซิงโครไนซ์สัญญาณนาฬิกาทางฝั่งส่งและรับให้เท่ากันเพื่อให้รับ-ส่ง ข้อมูล

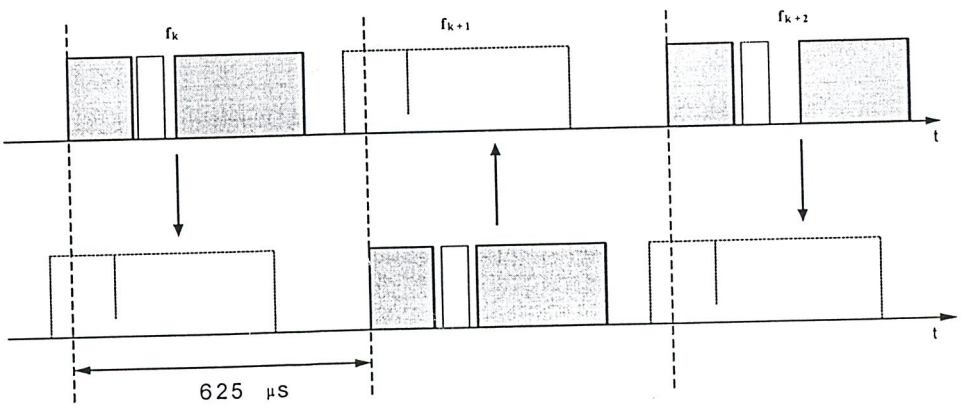
กันได้ถูกต้อง ซึ่งเมื่อรวมปริมาณข้อมูลทั้งหมดที่จำเป็นต้องส่งในแต่ละครั้งจะทำให้ความเร็วในการส่งข้อมูลจริงลดลงจาก 1 เมกะบิตต่อวินาทีเหลือ 723.2 kbit/sในทิศทางหนึ่งและ 57.6 kbit/sในอีกทิศทางหนึ่ง

นอกจากความเร็วและความสะดวกสบายแล้ว สิ่งที่จำเป็นอย่างยิ่งสำหรับการสื่อสารข้อมูลในปัจจุบันก็คือ ปลอดภัยของข้อมูล โดยเฉพาะอุปกรณ์บลูทูธ ที่สามารถทำงานได้ในทุกที่ยิ่งมีความจำเป็นที่จะต้องมีการรักษาความปลอดภัยของข้อมูลเป็นอย่างดี เทคนิคการส่งข้อมูลที่บลูทูธ ใช้คือ เทคนิคการกระโดดข้ามทางความถี่ (Frequency Hopping Spread Spectrum : FHSS)

2.7 การกระโดดทางความถี่ (Frequency Hopping)

เทคนิค FHSS ที่บลูทูธ ใช้นี้จะแบ่งข้อมูลที่ต้องการส่งออกเป็นแพ็กเกจ การส่งข้อมูลในแพ็กเกจแรกจะเลือกความถี่ของช่องสัญญาณช่องหนึ่งสำหรับการส่ง หลังจากส่งเสร็จสิ้นก็จะกระโดดไปเลือกใช้ช่องสัญญาณความถี่อื่นในการส่งแพ็กเกจที่สอง และจะกระโดดไปใช้ความถี่อื่นเรื่อย ๆ ตลอดช่วงความถี่ที่สามารถใช้งานได้ การกระโดดไปใช้งานช่องความถี่ต่าง ๆนี้เรียกว่า Hopping มาสเตอร์จะเป็นตัวกำหนด Frequency hopping sequence โดยที่สเลฟจะทำงานไปพร้อม ๆ กับมาสเตอร์ โดยการทำงานของสเลฟจะทำงานตามหลัง Master's hopping sequency

การจัดช่องสัญญาณของ บลูทูธ นั้นจะใช้วิธีการแบบ Frequency Hop/Time division Duplex (FH/TDD) ดังรูปที่ 2.5 ช่องสัญญาณถูกแบ่งทางเวลาออกเป็นช่วงเวลาละ 625 μs ที่เรียกว่า สล็อท(slot) แต่ละสล็อทจะใช้ความถี่ในการกระโดดที่แตกต่างกัน อัตราการกระโดดเปลี่ยนความถี่ จะเท่ากับ 1,600 hops/sec แพ็กเกจเดียวกันเท่านั้นที่สามารถถูกส่งต่อ 1 ช่องสล็อท ช่องสล็อทที่ทำการส่งและการรับจะวางเรียงสลับกันไปในลักษณะที่เรียกว่า Time Division Duplex (TDD)



รูปที่ 2.5 Frequency hop / Time division duplex channel

ในทุก ๆ อุปกรณ์ บลูทูธ จะมีแอดเดรส (address) และค่าคล็อก (clock) เฉพาะตัวอยู่ เมื่อสเลฟทำการเชื่อมต่อไปยังมาสเตอร์ อุปกรณ์ที่เป็นสเลฟจะถูกกำหนดค่าแอดเดรสและคล็อกที่เป็นของมาสเตอร์ จากนั้นจะนำค่าเหล่านี้ไปใช้ในการคำนวณ Frequency hopping sequence เนื่องจากอุปกรณ์ที่เป็นสเลฟจะต้องใช้แอดเดรส และ คล็อก เดียวกันกับอุปกรณ์ที่เป็นมาสเตอร์ และจะทำงานไปพร้อมกันกับ Master's hopping sequency การส่งข้อมูลทำได้โดยขั้นแรก มาสเตอร์จะอนุญาตให้ทำการส่งสัญญาณได้ โดยการกำหนดสล็อท (slot) สำหรับการส่งสัญญาณ ซึ่งมีอยู่สองชนิดคือ Voice traffic และ Data traffic ใน Data traffic อุปกรณ์ที่เป็นสเลฟจะทำการส่งข้อมูลกันได้ก็ต่อเมื่อได้รับสัญญาณจากอุปกรณ์ที่เป็นมาสเตอร์เท่านั้น และใน Voice traffic อุปกรณ์ที่เป็นสเลฟจะสามารถส่งข้อมูลได้โดยไม่ต้องรอสัญญาณจากอุปกรณ์ที่เป็นมาสเตอร์

จุดเด่นของการใช้เทคนิคนี้ในการส่งสัญญาณมีอยู่ 2 ข้อ คือ

- เกิดการชนกันของการเลือกใช้ช่องสัญญาณน้อย เนื่องจากช่วงความถี่ ISM ที่ บลูทูธ ใช้นั่นเป็นช่วงความถี่ที่ไม่ต้องขออนุญาต ทำให้มีอุปกรณ์หลายชนิดที่ใช้ความถี่ช่วงนี้อยู่ ประกอบกับรูปแบบการใช้งานอุปกรณ์ Bluetooth ส่วนใหญ่จะอยู่ในลักษณะที่สามารถเคลื่อนที่ไปใช้งานที่ตำแหน่งใด ๆ ก็ได้ ดังนั้นจึงมีความเสี่ยงที่จะเกิดการใช้ช่องสัญญาณที่ซ้ำกันได้ ถ้าอุปกรณ์เลือกจับช่องสัญญาณใด ๆ สำหรับส่งสัญญาณแบบไม่เปลี่ยนช่อง แต่ถ้าใช้เทคนิค FHSS โดยที่กำหนดช่วงเวลาในการจับช่องสัญญาณของการส่งข้อมูลแต่ละครั้งให้สั้น ก็จะทำให้โอกาสที่จะเกิดการใช้งานช่องความถี่เดียวกันลดลง และถึงแม้จะเกิดการชนกันของข้อมูลไปเพียงแพ็กเกจเดียว เมื่อส่งข้อมูลซ้ำในครั้งถัดไป (Retransmit) ก็จะเปลี่ยนไปใช้ความถี่อื่นซึ่งโอกาสที่จะไปใช้ช่องความถี่ซ้ำกันอีกมีได้น้อยมากเพราะรูปแบบในการกระโดดของอุปกรณ์แต่ละตัวจะไม่เหมือนกัน ความเร็วในการกระโดดที่บลูทูธ กำหนดไว้อยู่ที่ 1,600 ครั้งต่อวินาที
- มีความปลอดภัยของข้อมูลสูง ดังที่กล่าวไปแล้วว่าอุปกรณ์ที่เป็นมาสเตอร์จะคอยควบคุมจังหวะการรับส่งข้อมูลทั้งหมดนั่นหมายความรวมถึงรูปแบบการกระโดดเปลี่ยนช่องสัญญาณด้วย โดยรูปแบบการกระโดดนี้จะกำหนดจากแอดเดรสของอุปกรณ์ที่เป็นมาสเตอร์ ซึ่งแอดเดรสนี้จะไม่มีทางซ้ำกันเลยในอุปกรณ์ทุก ๆ ตัว นั่นหมายความว่าอุปกรณ์ที่เป็นสเลฟเท่านั้นที่จะรู้แอดเดรสของมาสเตอร์เพื่อไปคำนวณรูปแบบการกระโดดที่ถูกต้องเพื่อรับข้อมูลแต่ละแพ็กเกจในลำดับที่ถูกต้องแล้วประกอบขึ้นใหม่ให้เหมือนกับข้อมูลที่ส่งมา จากหลักการที่กล่าวมาจะเห็นได้ว่าเป็นการยากที่จะคาดเดารูปแบบการกระโดดที่ถูกต้องเพื่อรับข้อมูลได้ เทคนิคนี้ถูกคิดค้นขึ้นเพื่อนำมาใช้สำหรับรักษาความปลอดภัยของข้อมูลที่ใช้ควบคุมตอร์ปิโดในสมัยสงครามโลกครั้งที่ 2

ช่วงความถี่	ISM Band 2.400-2.4835 GHz	ยกเว้นฝรั่งเศสที่อยู่ในช่วง 2.4465-2.4835 GHz
วิธีการมอดูเลท	Gaussian Frequency-Shift Keying (GFSK)	BT Product = 0.5, Modulation Index= 0.28 – 0.35
ความเร็วในการส่งข้อมูล	1 Mbit/sec	การมอดูเลทด้วยวิธี GFSK ทำให้สามารถส่งข้อมูลได้ 1 บิตต่อความถี่พาหะ 1 เฮิรตซ์
ความเร็วที่ใช้ส่งข้อมูลได้จริง	723.2 kbit/sec ในทิศทางหนึ่งและ 57.6 kbit/sec ในอีกทิศทางหนึ่ง	เกิดจากความเร็วทั้งหมด 1 Mbit/sec แล้วตัดโอเวอร์เฮดต่าง ๆ ที่ต้องใช้ในการสื่อสาร
ความถี่ในการกระโดดเปลี่ยนช่อง	1,600 ครั้งต่อวินาที	มีเวลา 625 μs ต่อการกระโดด 1 ครั้ง
ความไวของอุปกรณ์ตัวรับ	ต้องมีค่า Bit Error Rate (BER) ที่ดีกว่า 0.1% ที่ระดับความแรงของสัญญาณอินพุต –70dBm หรือน้อยกว่า	ความไวที่ –70 dBm นี้รับสัญญาณจากตัวส่งของอุปกรณ์ใด ๆ ที่ตรงตามมาตรฐาน Bluetooth
ความแรงของสัญญาณที่ส่ง	แบ่งเป็น 3 คลาสคือ 0 dBm, 4dBm และ 20dBm	ระยะทำงานขึ้นอยู่กับความแรงของสัญญาณ โดยที่คลาส 3 มีระยะทำงานอยู่ในช่วง 5-10 และคลาส 1 ใช้ได้ไกลสุด 100 เมตร

ตารางที่ 2.3 สรุปข้อมูลต่างๆ ของเทคโนโลยีบลูทูธ

2.8 ระบบการรักษาความปลอดภัยของข้อมูลบลูทูธ (Bluetooth Security)

ในการส่งสัญญาณระหว่างอุปกรณ์บลูทูธนั้น มีระบบรักษาความปลอดภัยเพื่อป้องกันการรุกล้ำเข้ามาล้วงเอาข้อมูล(hack) โดยแบ่งการทำงานออกเป็น 3 หมวด ดังนี้

(1) ไม่ใช้ระบบรักษาความปลอดภัย (Unsecured) คือไม่มีการป้องกันการล้วงเอาข้อมูล ผู้ใช้ที่มีอุปกรณ์บลูทูธ สามารถเข้ามาเชื่อมต่อถึงกันในเครือข่ายได้ทันทีเมื่อเข้ามาในขอบเขตรัศมีการติดต่อ

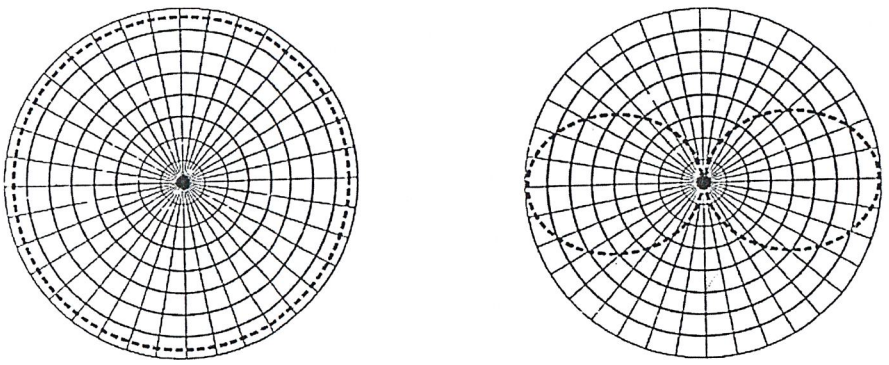
(2) ระบบรักษาความปลอดภัยแบบเซอร์วิส (Service secure) คือมีการแสดงตัวผู้ใช้(user) ก่อนการเข้าถึงหรือเชื่อมต่อระหว่างกัน เพื่อเป็นการยืนยันว่าเป็นผู้ที่ได้รับการอนุญาตให้เข้ามาเชื่อมต่อได้

(3) ระบบรักษาความปลอดภัยแบบลิงค์ (Link secure) คือมีการแสดงตัวผู้ใช้เช่นเดียวกับ Service secure แต่เพิ่มความปลอดภัยมากขึ้นด้วยการเข้ารหัสข้อมูลที่จะทำการส่งถึงกันก่อนที่จะทำการ

ส่งสัญญาณ ซึ่งถ้าหากมีการล้วงเอาข้อมูลระหว่างที่ทำการส่งสัญญาณนั้น ผู้ล้วงเอาข้อมูลก็จะไม่สามารถเข้าใจข้อมูลที่ล้วงเอาไปได้ เนื่องจากมีการทำการเข้ารหัสเอาไว้ก่อนแล้ว

2.9 สายอากาศ

รูปแบบการแพร่กระจายคลื่นของสายอากาศมักจะถูกพล็อตเป็น 2 มิติ ก็คือแกน azimuth และ elevation รูปแบบของ azimuth ต้องมองมาที่ตัวสายอากาศจากด้านบน รูปแบบของ elevation ต้องมองมาที่ตัวสายอากาศจากด้านข้าง สายอากาศไดโพลจะแพร่กระจายคลื่นในรูปแบบฐานวงกลม รูปที่ 2.6 แสดงรูปแบบการแพร่กระจายคลื่น ทางซ้ายคือ azimuth การแพร่กระจายที่ดีที่สุด จะต้องเป็นรูปวงกลมซึ่งประกอบด้วยความเข้มของคลื่นที่เท่ากันทุกทิศทาง และรูปทางด้านขวาคือรูปแบบของ elevation ที่ทางด้านข้างของสายอากาศ การแพร่กระจายคลื่นจะมีความเข้มมากที่สุด และจะลดลงจนไม่มีความเข้มที่ด้านบนและด้านใต้ของสายอากาศ



รูปที่ 2.6 รูปแบบการแพร่กระจายคลื่น

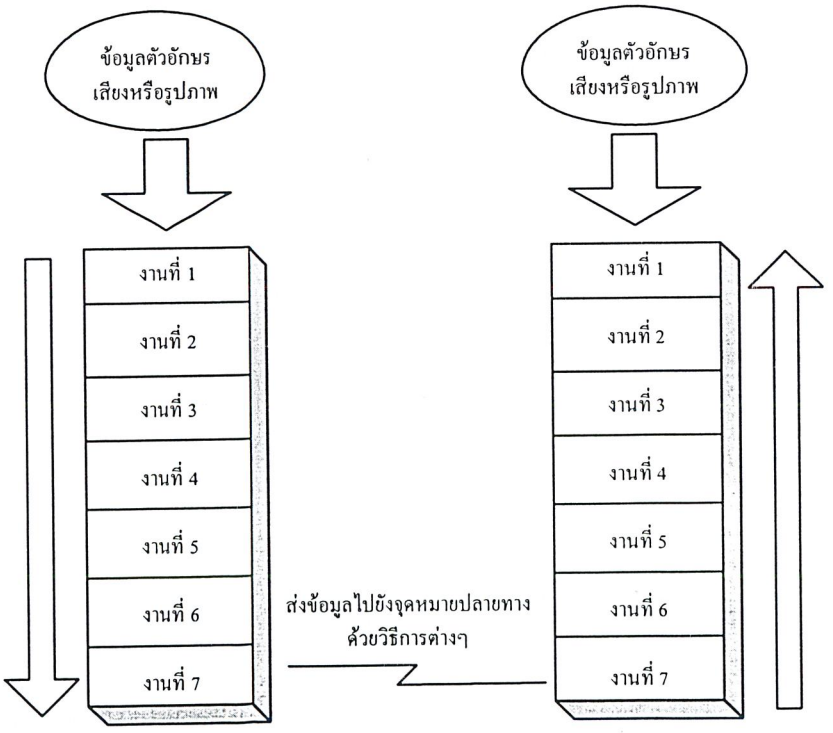
รูปแบบการแพร่กระจายคลื่นของสายอากาศถูกใช้ในการออกแบบของ บลูทูธ เพราะว่ามันจะบอกเราว่าความแรงของสัญญาณจะแตกต่างกันเมื่อมองในมุมที่ต่างกัน ดังเช่นสายอากาศที่ยกขึ้นมา ด้านบนสายอากาศจะทำงานดีที่สุดเมื่อทำมุมที่ 90 องศา กับตัวอุปกรณ์ที่ต้องการจะติดต่อด้วย ในมุมอื่น ๆ สัญญาณก็จะมีความแรงเมื่อมุมมากกว่า 45 องศา ดังนั้นอุปกรณ์ควรจะต้องทำมุมกับสายอากาศมากกว่า 45 องศา จึงจะทำให้สายอากาศทำงาน ที่ที่ควรจะวางควรจะไม่มีสิ่งกีดขวางเพื่อให้สามารถติดต่อกันได้ดีและมีสัญญาณเพียงพอในการติดต่อ สายอากาศบางชนิดมีการแพร่กระจายคลื่นที่มีทิศทางแน่นอน และบางชนิดจะเข้าใกล้สู่การเป็นไอโซทรอปิก (isotropic) คือรูปแบบทรงที่มีสัญญาณทุกทิศทาง ความสามารถของสายอากาศนี้ความเข้มของการแพร่กระคลื่นในทิศทางเฉพาะ คือ ไดเรกทีฟเกน

2.10 โปรโตคอลสแตก

สำหรับอุปกรณ์ที่มีการเชื่อมต่อเข้าด้วยกันเป็นเน็ตเวิร์คนั้น การส่งข้อมูลจากอุปกรณ์หนึ่งไปยังไปอุปกรณ์ปลายทางจำเป็นต้องมีการส่งข้อมูลอื่น ๆ ประกอบเข้าไปกับข้อมูลที่ต้องการส่งนั้นด้วย เพื่อควบคุมเส้นทางของข้อมูลให้สามารถส่งไปอุปกรณ์ปลายทางได้อย่างถูกต้อง ทำให้การส่งข้อมูลแต่ละครั้งเกิดการทำงานต่าง ๆ ขึ้นมากมายจึงเกิดการสร้างโมเดลแทนการทำงานต่าง ๆ ที่ว่านี้ขึ้น เพื่อให้สามารถมองเห็นภาพรวมของการทำงานทั้งหมดได้

โมเดลนี้จะแบ่งออกเป็นชั้นหรือเลเยอร์ (Layer) ซ้อนกันในแนวตั้ง ในแต่ละชั้นจะมีขอบเขตหน้าที่การทำงานของตัวเองและเมื่อทำงานเสร็จก็จะส่งงานต่อไปให้ชั้นถัดไปทำงาน โดยชั้นบนสุดคือส่วนที่ติดต่อกับผู้ใช้งานและชั้นล่างสุดคือส่วนที่เกิดการส่งข้อมูลขึ้นจริง ทิศทางการเคลื่อนที่ของงานก็จะขึ้นอยู่กับทิศทางของการรับ-ส่งข้อมูล คือถ้าเป็นการส่งข้อมูลงานก็จะเริ่มจากผู้ใช้ส่งเข้ามายังชั้นบนสุดแล้วส่งต่อลงมาเรื่อย ๆ จนถึงชั้นล่างสุดแล้วส่งไปยังตัวรับที่ชั้นล่างสุด จากนั้นข้อมูลที่ตัวรับก็จะถูกส่งขึ้นไปเรื่อย ๆ จนถึงชั้นล่างสุดแล้วส่งไปยังตัวรับที่ชั้นล่างสุด จากนั้นข้อมูลที่ตัวรับก็จะถูกส่งขึ้นไปเรื่อย ๆ จนถึงชั้นบนสุดซึ่งผู้ใช้ทางฝั่งรับสามารถเห็นข้อมูลนี้เหมือนกับต้นฉบับทางฝั่งส่ง

การทำงานเป็นลำดับขั้นตอนนี้เรียกอีกอย่างหนึ่งว่า โปรโตคอลสแตก (Protocol Stack) ดังรูปที่ 2.7 แสดงให้เห็นการทำงานของโมเดลที่เป็นลำดับขั้น ระหว่างการส่งข้อมูลจากอุปกรณ์หนึ่งไปยังอีกอุปกรณ์หนึ่ง



รูปที่ 2.7 โมเดลการทำงานระหว่างการส่งข้อมูลจากอุปกรณ์หนึ่งไปยังอุปกรณ์หนึ่ง

ประโยชน์ของการสร้างโมเดลเป็นชั้น ๆ ที่มีขอบเขตหน้าที่การทำงานในแต่ละชั้นที่แน่นอนนั้นก็เพื่อความรวดเร็วในการพัฒนาอุปกรณ์ กล่าวคือนักพัฒนาสามารถพัฒนาอุปกรณ์เฉพาะส่วนในชั้นต่าง ๆ แยกจากกันได้โดยอิสระ ทำให้เกิดความรวดเร็วในการพัฒนาอุปกรณ์ต่าง ๆ

โดยได้มีการกำหนดโครงสร้างการทำงานของอุปกรณ์ขึ้นมาเป็นโมเดลมาตรฐาน เพื่อใช้เป็นแบบจำลองสำหรับอ้างอิงการทำงานของอุปกรณ์และโปรแกรมที่เชื่อมต่อกันเป็นเน็ตเวิร์ค โมเดลนี้เรียกว่าโมเดล OSI (OSI Model) ซึ่งมีการแบ่งการทำงานทั้งหมดออกเป็น 7 ชั้นด้วยกันดังรูปที่ 2.8 ขอบเขตหน้าที่การทำงานของชั้นต่าง ๆ เป็นดังนี้

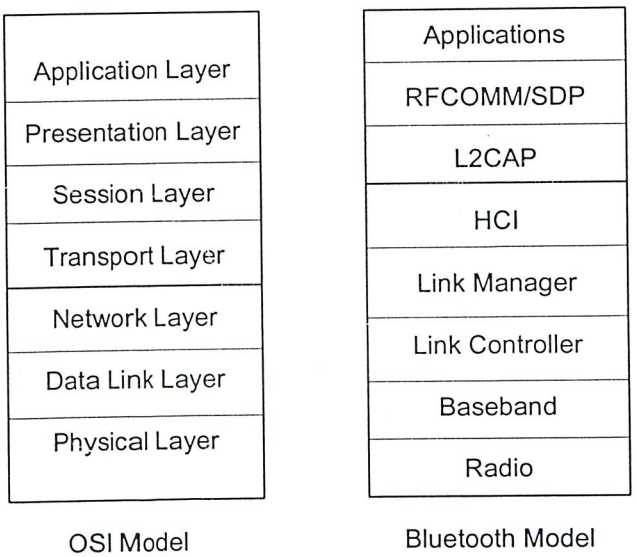
- **ชั้นที่ 7 Application Layer** เป็นส่วนของโปรแกรมที่ติดต่อรับหรือส่งข้อมูลกับผู้ใช้
- **ชั้นที่ 6 Presentation Layer** จะทำการแบ่งข้อมูลออกเป็นชนิดต่าง ๆ ตัวอักษร ภาพ หรือเสียง รวมถึงการเข้ารหัสข้อมูล
- **ชั้นที่ 5 Session Layer** ทำหน้าที่แบ่งข้อมูลที่ต้องการส่งทั้งหมดออกเป็นส่วนย่อยเพื่อส่งทีละส่วน
- **ชั้นที่ 4 Transport Layer** แปลงคำสั่งจากระดับบนไปสู่ชุดคำสั่งที่ระดับล่างรู้จัก
- **ชั้นที่ 3 Network Layer** ค้นหาเส้นทางที่จามารถส่งข้อมูลไปถึงที่หมาย
- **ชั้นที่ 2 Data Link Layer** นำข้อมูลและเส้นทางที่ได้มาสร้างเป็นเฟรมข้อมูลพร้อมที่จะส่ง
- **ชั้นที่ 1 Physical Layer** ทำหน้าที่ส่งเฟรมข้อมูลออกไปยังอุปกรณ์ปลายทางโดยผ่านสื่อชนิดต่าง ๆ ตามที่ระบบกำหนดไว้

จะเห็นได้ว่าถ้าอุปกรณ์ใดก็ตามที่มีการเชื่อมต่อกันเป็นเน็ตเวิร์คจะมีลำดับการทำงานคล้ายกับโมเดล OSI นี้แต่ในการกำหนดมาตรฐานของแต่ละระบบเน็ตเวิร์คก็จะมีการสร้างโมเดลของตัวเองขึ้นมาซึ่งจะมีลำดับการทำงานที่สอดคล้องกับโมเดล OSI จะแตกต่างก็เฉพาะจำนวนชั้นและขบเขตการทำงานของแต่ละชั้นที่ปรับเปลี่ยนให้เข้ากับระบบนั้น ๆ

สำหรับโมเดลการทำงานของ Bluetooth (Bluetooth Model) ถูกกำหนดให้มีโครงสร้างการทำงานดังรูปที่ 2.8 ซึ่งจะเห็นได้ว่ามีจำนวน 8 ชั้นมากกว่าโมเดล OSI อยู่ 1ชั้น ทำให้ขอบเขตการทำงานในแต่ละชั้นแตกต่างจากโมเดล OSI แต่ลำดับการทำงานมีลักษณะเหมือนกัน โดยแต่ละชั้นของโมเดล Bluetooth มีชื่อเรียกและหน้าที่การทำงานดังนี้

- **ชั้นที่ 8 Applications** เป็นส่วนของโปรแกรมที่ติดต่อรับหรือส่งข้อมูลกับผู้ใช้
- **ชั้นที่ 7 RFCOMM/SDP** สำหรับ RFCOMM เป็นโปรโตคอลเสมือนที่ทำให้แอพพลิเคชันด้านบนมอง Bluetooth เป็นเหมือนพอร์ตอนุกรม (Serial Port) ทั่วไป ส่วน SDP (Service Discovery Protocol) เป็นโปรโตคอลที่ช่วยค้นหาบริการจากอุปกรณ์ Bluetooth ตัวอื่นที่อยู่ในขอบเขตพิโคเน็ทเดียวกัน

- **ชั้นที่ 6 L2CAP** (Logical Link Control and Adaptation Protocol) ทำหน้าที่มัลติเพล็กซ์ข้อมูลจากชั้นบนซึ่งอาจจะมีการทำงานของโปรแกรมหลายโปรแกรมพร้อมกันและจัดแบ่งข้อมูลออกเป็นแพ็กเกจ
- **ชั้นที่ 5 HCI** (Host Controller Interface) เป็นโปรโตคอลเชื่อมต่อระหว่างโปรแกรมชั้นบนที่ทำงานอยู่บนระบบหนึ่ง (เช่นโปรแกรมในเครื่องคอมพิวเตอร์โน้ตบุ๊คทำงานบน CPU x86) กับส่วนควบคุมการทำงานของ Bluetooth (เช่นการ์ด PCMCIA Bluetooth ที่ต่ออยู่ในเครื่องคอมพิวเตอร์โน้ตบุ๊ค) ทำให้โปรแกรมรู้จักคำสั่งควบคุมอุปกรณ์ บลูทูธ
- **ชั้นที่ 4 Link Manager** ทำหน้าที่แปลงคำสั่งที่ได้รับจากชั้นบนเป็นลำดับหน้าที่การทำงานที่ชั้นล่างรู้จัก และคอยส่งคำสั่งลงไปควบคุมการทำงานของชั้นล่างทั้งหมด
- **ชั้นที่ 3 Link Controller** ควบคุมการเชื่อมต่อพื้นฐานของ Bluetooth ทั้งหมด ไม่ว่าจะเป็นสถานะของอุปกรณ์ โหมดการทำงานของอุปกรณ์ การค้นหาอุปกรณ์ Bluetooth ใกล้เคียง รวมไปจนถึงการเลือกว่าจะเป็นมาสเตอร์หรือสเลฟในสภาพแวดล้อมต่าง ๆ
- **ชั้นที่ 2 Baseband** การทำงานของชั้นนี้ถือได้ว่าเป็นหัวใจของ Bluetooth ในด้านฮาร์ดแวร์เลยก็ว่าได้ หน้าที่หลักของชั้นนี้ คือการควบคุมวงจรภาคส่ง-รับคลื่นวิทยุที่อยู่ในชั้นล่างสุด ซึ่งจุดสำคัญที่สุดของการควบคุมก็คือการเลือกช่องความถี่ในการรับส่งข้อมูลให้ตรงกันระหว่างมาสเตอร์และสเลฟ ที่ต้องมีการกระโดดไปในรูปแบบเดียวกัน
- **ชั้นที่ 1 Radio** เป็นส่วนที่เกิดการส่งและรับคลื่นวิทยุจริง ๆ เป็นส่วนฮาร์ดแวร์วงจรภาคส่ง-รับคลื่นวิทยุที่ถูกควบคุมจากชั้น Baseband ไม่ว่าจะเป็นความถี่และระดับความแรงของสัญญาณที่ใช้ รวมไปถึงเฟรมข้อมูลที่จะส่ง



รูปที่ 2.8 การเปรียบเทียบระหว่างโมเดลมาตรฐาน OSI กับโมเดลของบลูทูธ

2.11 ภาครับ-ส่งสัญญาณวิทยุและสายอากาศ (Radio)

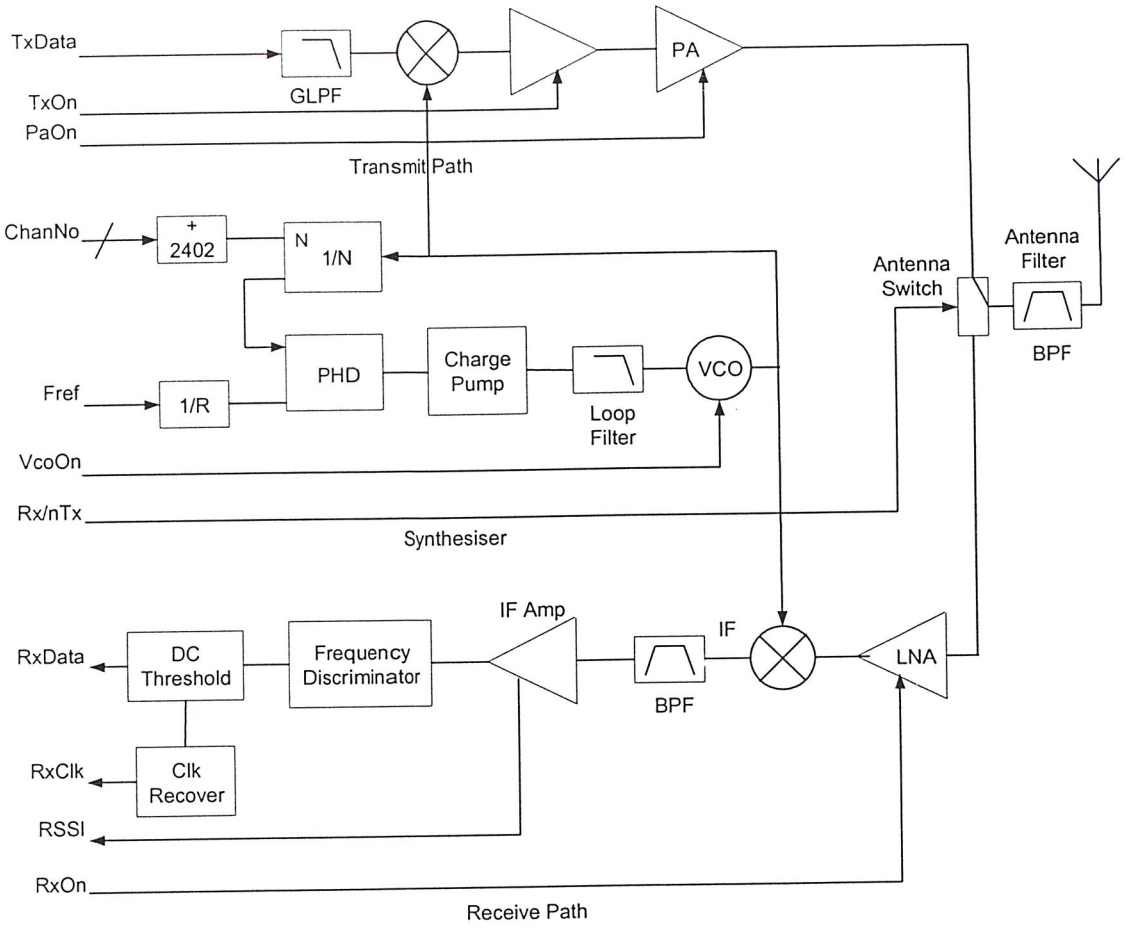
หน้าที่พื้นฐานของภาคนี้ก็คือการรับและส่งสัญญาณวิทยุที่ผ่านการมอดูเลทข้อมูลเข้าไปแล้ว รูปที่ 2.9 แสดงให้เห็นบล็อกไดอะแกรมอย่างง่ายของโครงสร้างการทำงานภาครับ-ส่งสัญญาณวิทยุ

จากรูปจะเห็นได้ว่าสามารถแบ่งการทำงานทั้งหมดออกได้เป็น 3 ส่วน คือ ส่วนมอดูเลทข้อมูลภาคส่ง (บนสุด) ส่วนดีมอดูเลทข้อมูลภาครับ (ล่างสุด) และส่วนสร้างความถี่พาหะ (กลาง) จะเห็นได้ว่าแต่ละส่วนจะมีการรับ-ส่งข้อมูลไปยังภายนอกซึ่งส่วนที่มาเชื่อมต่อนี้ก็คือภาคเบสแบนด์ (Baseband) นั้นเอง

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

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

ในส่วนของการสร้างความถี่พาหะจะได้รับการเลือกช่องสัญญาณจากภาคเบสแบนด์ จากนั้นจะนำหมายเลขของช่องสัญญาณไปบวกด้วย 2,402 เมกะเฮิรตซ์ก็จะได้ค่าความถี่ที่ต้องการ จากนั้นก็จะนำค่าที่ได้นี้ไปป้อนให้แก่วงจรเฟสล็อกลูป (Phased- Lock Loop) เพื่อสร้างความถี่ที่ต้องการออกมา



รูปที่ 2.9 โครงสร้างของภาครับส่งสัญญาณวิทยุ

2.12 ภาคเบสแบนด์ (Baseband)

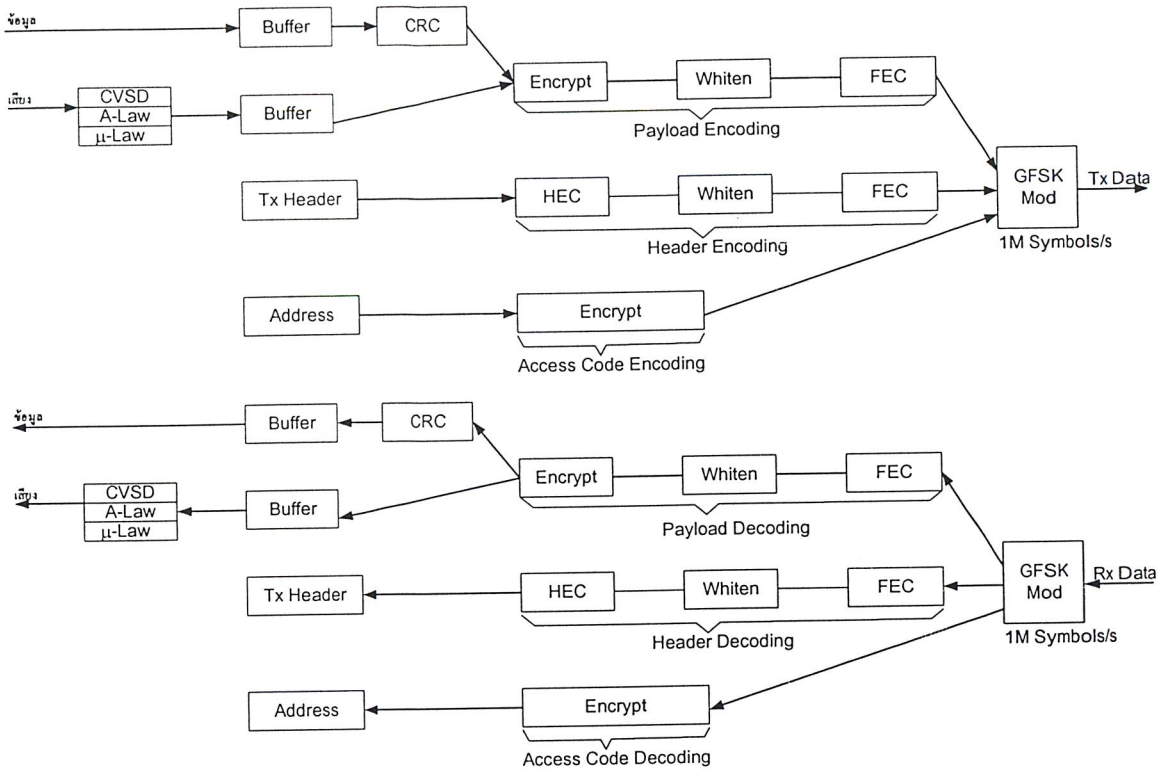
การทำงานทั้งหมดของภาครับ – ส่งสัญญาณวิทยุ จะถูกควบคุมโดยภาคเบสแบนด์ทั้งในด้านอัตราขยายสัญญาณวิทยุและช่องความถี่ที่ใช้งาน นอกจากการควบคุมการทำงานของภาครับ-ส่งสัญญาณวิทยุแล้วอีกหน้าที่หนึ่งที่สำคัญก็คือการสร้าง Access Code, Header และส่วนตรวจสอบความผิดพลาด รวมเข้ากับข้อมูลที่ต้องการส่งจริงเพื่อสร้างเป็นแพ็กเกจ

อัตราขยายของวงจรส่งจะถูกกำหนดให้มีค่าสูงสุด (ตามคลาสที่กำหนด) ในการส่งครั้งแรก แต่เมื่อมีการเชื่อมต่อกันสมบูรณ์แล้วอุปกรณ์ฝั่งรับจะส่งค่าความแรงของสัญญาณที่รับกลับมา ซึ่งอุปกรณ์ฝั่งส่งจะนำค่าที่ได้รับกลับมานี้ไปปรับค่าอัตราขยายไม่ให้สูงเกินจำเป็นเพื่อประหยัดพลังงาน ในส่วนของการเลือกช่องความถี่ในการส่งหรือรับนั้นจะคำนวณจากค่าแอดเดรสของอุปกรณ์ที่เป็นมาสเตอร์ในแต่ละพิโคเน็ท ซึ่งค่าแอดเดรสนี้จะถูกส่งออกมาจากมาสเตอร์ในส่วนของ Access Code ในทุก ๆ แพ็กเกจข้อมูล นั่นหมายความว่าอุปกรณ์ทุกตัวที่อยู่ในพิโคเน็ท (เป็นมาสเตอร์-สเลฟกัน) จะสามารถคำนวณเลือกช่องความถี่ที่เหมือนกันทำให้

สามารถรับ-ส่งข้อมูลกันได้อย่างถูกต้อง ในขณะที่อุปกรณ์ในพิโคเน็ทอื่นก็จะมีรูปแบบการกระโดดที่แตกต่างออกไปเพราะค่าแอดแดรสของอุปกรณ์ที่เป็นมาสเตอร์ไม่เหมือนกัน (อุปกรณ์บลูทูธทุกตัวจะมีแอดแดรสของตัวเองที่ไม่ซ้ำกันเลย)

รูปที่ 2.10 แสดงให้เห็นถึงกระบวนการสร้างแพ็กเกจข้อมูลของภาคเบสแบนด์ซึ่งจะเห็นได้ว่าแบ่งออกได้เป็น 3 ส่วนใหญ่ๆคือ ส่วนของ Access Code, Header และข้อมูลจริง (Payload Data) โดยในขั้นตอนการสร้างแพ็กเกจข้อมูลจะมีบล็อกการทำงานย่อยๆได้แก่

- 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 มีอยู่สองแบบคือ แบบ 1/3 และ 2/3 โดยการเข้ารหัสแบบ 1/3 จะมีความแข็งแรงต่อความผิดพลาดมากกว่าแบบ 2/3 แต่ก็ต้องเพิ่มพาริตีบิตเป็นจำนวนมากกว่า ใช้กับข้อมูลที่มีความสำคัญมาก เช่น Header เท่านั้น ส่วนข้อมูลทั่วไปถ้าจะเข้ารหัสจะใช้แบบ 2/3 เท่านั้น



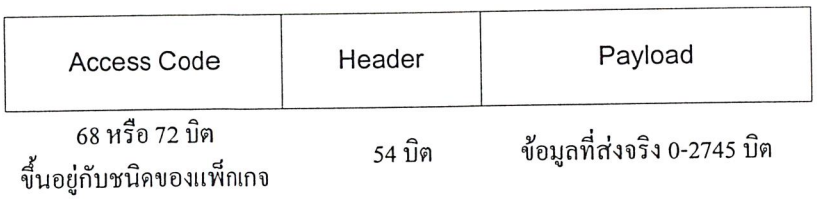
รูปที่ 2.10 การสร้างแพ็กเกจข้อมูลในภาคเบสแบนด์

ส่วน Access Code เกิดจากการนำแอดเดรสของอุปกรณ์มาผ่านการเข้ารหัสตามที่ระบุไว้ในมาตรฐานบลูทูธ มาสเตอร์หรือสเลฟที่ทำการรับข้อมูลบนเครือข่ายพิโคเน็ท จะทำการเปรียบเทียบสัญญาณที่เข้ามากับค่า access code ถ้า access code ไม่ตรงกัน ก็จะไม่พิจารณาแพ็กเกจที่รับเข้ามา ประโยชน์ของ access code นั้นนอกเหนือจากการใช้บ่งบอกเพื่อแยกแยะแพ็กเกจแล้ว ยังถูกใช้สำหรับการซิงโครไนซ์และการชดเชยสำหรับ offset อีกด้วย รหัส access code นี้ต้องมีคุณสมบัติที่สามารถทนทานต่อสัญญาณแทรกแซงที่เข้ามารบกวนได้

ส่วนของ Header เป็นส่วนที่จะเก็บข้อมูลการควบคุมที่สำคัญ เช่น Media Access Control(MAC) address , packet type , flow control bits , บิตที่ใช้สำหรับ ARQ (Automatic Retransmission Query) และฟิลด์ของ Header Error Check(HEC) Header ซึ่งมีความยาว 54 บิต เกิดจาการนำข้อมูลบ่งบอกลักษณะของแพ็กเกจและข้อมูลอื่นๆ ที่จำเป็นมาคำนวณส่วนป้องกันความผิดพลาดทั้งแบบ CRC และ FEC

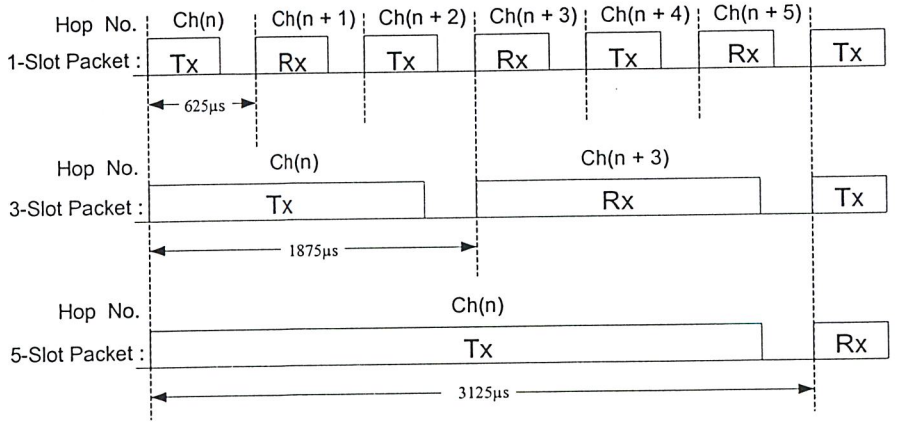
ส่วนของข้อมูล(playload) อาจจะมีหรือไม่มีก็ได้ จะสามารถแบ่งออกได้เป็น 2 ชนิด คือข้อมูลที่เป็นเสียงกับข้อมูลทั่วไป ซึ่งข้อมูลทั้งสองชนิดจะมีกระบวนการสร้างแพ็กเกจแตกต่างกัน 1 ขั้นตอน คือถ้าเป็นข้อมูลทั่วไปตัวข้อมูลจะถูกนำไปคำนวณส่วนตรวจสอบความผิดพลาด CRC-16 ในขณะที่ข้อมูลเสียงจะไม่มี เพราะในการส่งข้อมูลเสียงจะเป็นการส่งแบบเรียลไทม์ ไม่สามารถส่งข้อมูลซ้ำได้ถึงแม้จะเกิดการผิดพลาดจึงไม่มีส่วนตรวจสอบนี้

ข้อมูลทั้ง 3 ส่วนเมื่อผ่านกระบวนการทั้งหมดแล้วก็จะถูกนำมารวมเพื่อสร้างเป็นแพ็กเกจ โดยส่วนแรกของแพ็กเกจคือ Access Code ตามมาด้วย Header และปิดท้ายด้วย ตัวข้อมูลที่จะส่งจริงๆ รูปที่ 2.11 แสดงให้เห็นส่วนประกอบของข้อมูลใน 1 แพ็กเกจ



รูปที่ 2.11 ส่วนประกอบของแพ็กเกจข้อมูล

ถ้าหากต้องการส่งอัตราเร็วบิตข้อมูลที่สูง แพ็กเกจแบบ มัลติสลอท(multi slot) จะถูกใช้ นั่นคือแพ็กเกจหนึ่งจะส่งด้วยการใช้ 1 สลอท, 3 สลอท หรือ 5 สลอท แต่แพ็กเกจหนึ่งต้องส่งด้วยการใช้ความถี่พาหะ(carrier) เดียวเสมอ ตัวอย่างเช่น ถ้า 4 สลอท ที่เรียงต่อกันคือ 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 ก็จะส่งแพ็กเกจนั้นด้วยความถี่ f_k แพ็กเกจถัดไปก็จะเริ่มต้นใน slot k+3 และใช้ความถี่ f_{k+3} ดังรูปที่ 2.12 แสดงแพ็กเกจข้อมูลแบบมัลติสลอท



รูปที่ 2.12 multi slot แบบต่าง ๆ

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 และใช้ในการส่งเสียงด้วยอัตราเร็ว 64 kbit/s เสียงที่ส่งจะไม่ถูกป้องกันความผิดพลาด แต่ถ้าช่วงเวลาห่างในการส่งแพ็กเกจแต่ละครั้งถูกลดลง การแก้ไขความผิดพลาดโดย FEC แบบ 2/3 หรือ 1/3 สามารถถูกเลือกนำมาใช้
- **การเชื่อมโยงแบบ ACL link** เป็นการเชื่อมโยงจากจุดหนึ่งไปยังหลายจุด(point to multipoint) ที่อาศัยการสวิตช์แพ็กเกจ (packet switched) และมีการส่งข้อมูลที่อาจจะสมมาตรกัน(symmetrical) หรือไม่สมมาตรกัน (asymmetrical) ใน 2 ทิศทางก็ได้ ปกติมักจะใช้สำหรับการส่งข้อมูลที่เป็นช่วง ๆ (bursty data transmission) อุปกรณ์ที่เป็นมาสเตอร์จะใช้วิธีการโพล (polling) ในการเชื่อมโยงการควบคุมแบบ ACL ซึ่งแพ็กเกจข้อมูลอาจจะเป็นแบบ 1 slot , 3 slot หรือ 5 slot แพ็กเกจข้อมูลที่ส่งอาจจะไม่ถูกป้องกันความผิดพลาด หรือถูกป้องกันด้วย FEC แบบ 2/3 อัตราเร็วข้อมูลสูงสุด 723.2 kbit/s ในทิศทางหนึ่ง และ 57.6 kbit/s ในทิศทางกลับกัน สามารถได้รับจากการใช้แพ็กเกจแบบ 5 slot ที่ไม่ถูกป้องกันความผิดพลาด ตารางที่ 2.4 เป็นการสรุปอัตราเร็วบิตข้อมูลที่สามารถได้รับจาก ACL link โดย DMx แทนแพ็กเกจข้อมูลแบบ x slot ที่เข้ารหัส FEC ส่วน DHx จะแทนแพ็กเกจข้อมูลที่ไม่ถูกป้องกันความผิดพลาด

Type	Symmetric(kbit/s)	Asymmetric(kbit/s)	
DM1	108.8	108.8	108.8
DH1	172.8	172.8	172.8
DM2	256.0	384.0	54.4
DH2	384.4	576.0	86.4
DM3	286.7	477.8	36.3
DH3	432.6	723.2	57.6

ตารางที่ 2.4 อัตราเร็วบิตข้อมูลที่สามารถได้รับบน ACL link

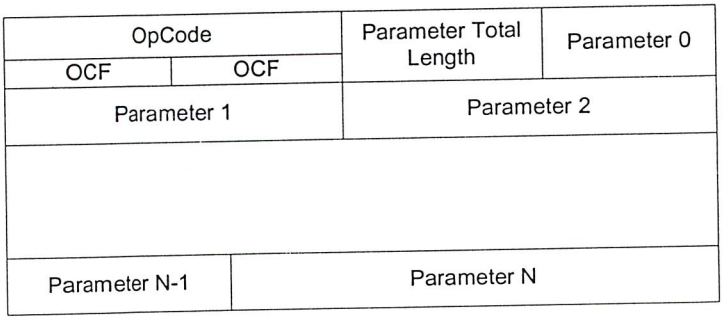
2.13 HCI (Host Controller Interface)

เป็นโปรโตคอลเชื่อมต่อระหว่างโปรแกรมชั้นบนที่ทำงานอยู่บนระบบหนึ่ง (เช่นโปรแกรมในเครื่องคอมพิวเตอร์โน้ตบุ๊คทำงานบน CPU x86) กับส่วนควบคุมการทำงานของ Bluetooth (เช่นการ์ด PCMCIA Bluetooth ที่ต่ออยู่ในเครื่องคอมพิวเตอร์โน้ตบุ๊ค) ทำให้โปรแกรมรู้จักคำสั่งควบคุมอุปกรณ์ บลูทูธ

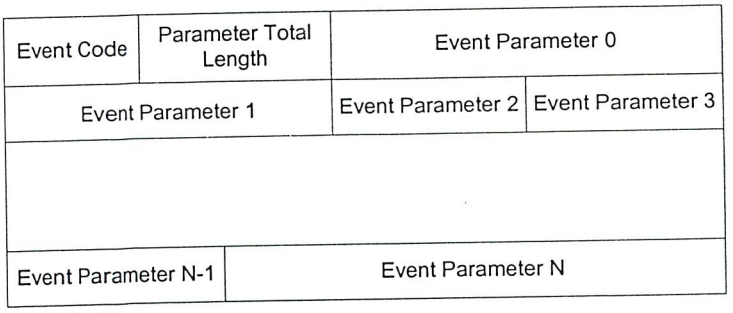
2.13.1 ชนิดของ HCI

บลูทูธได้แบ่งชนิดของ HCI ออกเป็น 3 แบบ ดังนี้

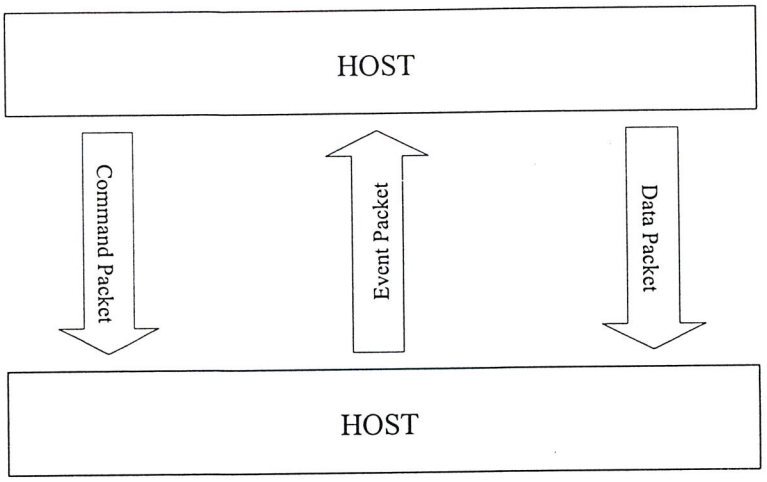
- HCI Command Packet ที่ถูกสร้างขึ้นโดยอุปกรณ์ควบคุม (Host) เพื่อนำไปควบคุมอุปกรณ์บลูทูธ (Host Controller) รูปแบบของ HCI Command Packet แสดงในรูปที่2.13 เมือ อุปกรณ์บลูทูธ แต่ละคำสั่งจะกำหนด OpCode มา 2 ไบต์ แบ่งเป็น OpCode Group Field (OGF) และ OpCode Command Field (OCF) โดย OGF จะจองพื้นที่มากสุด 6 บิท ส่วน OCF จะจองพื้นที่ 10 บิท
- HCI Event Packet ถูกสร้างโดยอุปกรณ์ บลูทูธ เพื่อแจ้งให้โฮสท์ทราบถึงความเปลี่ยนแปลงที่เกิดขึ้นในอุปกรณ์บลูทูธ ซึ่งอุปกรณ์บลูทูธต้องสามารถรองรับ HCI Event Packet ได้ 255 ไบต์ โดยยกเว้นส่วนของ HCI Event Packet Header รูปแบบของ HCI Event Packet แสดงในรูปที่ 2.14
- HCI Data Packet ทำการสื่อสารข้อมูล หรือเสียงระหว่างอุปกรณ์บลูทูธและอุปกรณ์ควบคุม แบ่งเป็น HCI ACL Data Packet และ HCI SCO Data Packet



รูปที่ 2.13 HCI Command Packet



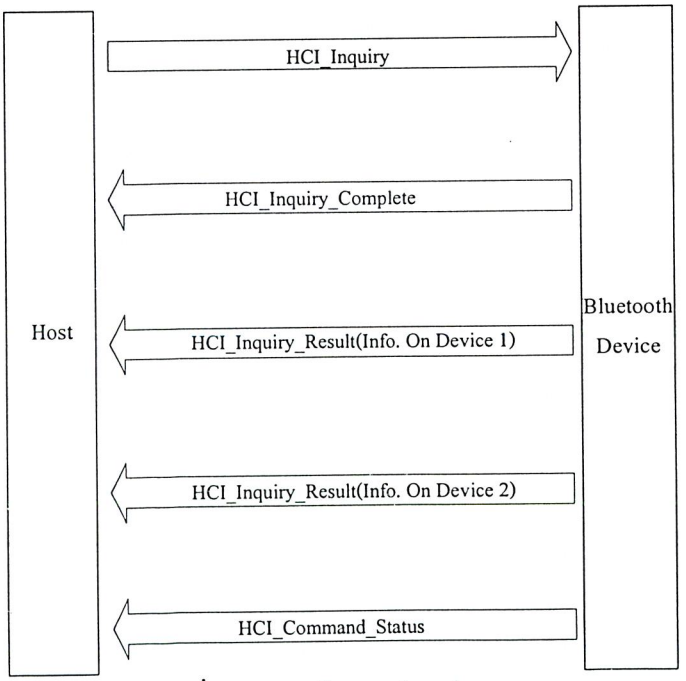
รูปที่ 2.14 HCI Event Packet



รูปที่ 2.15 HCI Command Packet

2.13.2 การค้นหาอุปกรณ์บลูทูธที่อยู่ใกล้เคียง

การค้นหาอุปกรณ์ที่อยู่ใกล้เคียง จะตอบสนองการรับ และการแสดงอุปกรณ์ โดยใช้คำสั่ง Host_Inquiry_Result ซึ่งในแต่ละ Host_Inquiry_Result ประกอบไปด้วยข้อมูลหลักคือ จำนวนของการตอบสนอง หรืออาจหมายถึงจำนวนของอุปกรณ์ บลูทูธ การทำงานของการค้นหาอุปกรณ์เป็นดังรูปที่ 2.16

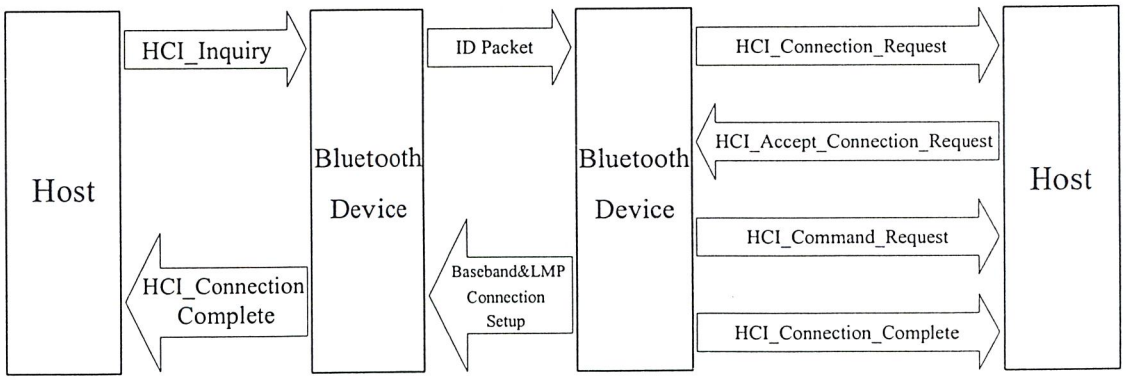


รูปที่ 2.16 การค้นหาอุปกรณ์บลูทูธ

โฮสท์จะ ทำการส่งข้อมูลการร้องขอการค้นหาอุปกรณ์ ไปยังอุปกรณ์ บลูทูธ (HCI_Inquiry) จากนั้นอุปกรณ์บลูทูธ จะทำการค้นหาอุปกรณ์ บลูทูธ ที่มีอยู่ใกล้เคียง และทำการแสดงผล (HCI_Inquiry_Result) ของอุปกรณ์ต่าง ๆ ไปยังโฮสท์ เมื่อทำการค้นหาอุปกรณ์ครบทั้งหมดแล้ว จะส่งข้อมูลแสดงการเสร็จสิ้นการค้นหาไปยังโฮสท์ (HCI_Inquiry_Complete)

2.13.3 การเชื่อมต่อกับอุปกรณ์ บลูทูธ

การเชื่อมต่อกับอุปกรณ์บลูทูธ ทำได้โดย โฮสท์ทำการส่งคำสั่ง HCI_Create_Connection เพื่อทำการระบุข้อมูลที่ต้องใช้ในการเชื่อมต่อ โดยมีข้อมูลหลักดังนี้ แอดเดรสของอุปกรณ์ที่จะทำการเชื่อมต่อ และหน้าที่โหมดการทำงานของอุปกรณ์ที่จะทำการเชื่อมต่อ(มาสเตอร์ หรือสเลฟ) จากนั้น อุปกรณ์ บลูทูธ ที่จะทำการถูกเชื่อมต่อจะได้รับHCI_Connection_request เพื่อแจ้งให้ทราบถึงการร้องขอการเชื่อมต่อ และจะทำการตอบรับ หรือ ปฏิเสธการเชื่อมต่อ ถ้าอุปกรณ์บลูทูธ ที่ถูกร้องขอปฏิเสธการเชื่อมต่อจะส่งคำสั่ง HCI_Reject_Connection_request ไปยังอุปกรณ์ บลูทูธ ที่ร้องขอการเชื่อมต่อ แต่ถ้าอุปกรณ์ บลูทูธ ที่ถูกร้องขอ ตอบรับการเชื่อมต่อ จะส่งคำสั่งโฮสท์ ของอุปกรณ์บลูทูธ ที่ถูกร้องขอจะส่งคำสั่ง HCI_Accept_Connect_request ไปยังอุปกรณ์บลูทูธ เพื่อทำการเซ็ตอัพ การเชื่อมต่อจากนั้นอุปกรณ์บลูทูธ ทั้งสองฝั่ง จะทำการส่งคำสั่ง (HCI_ Connection _Complete) ไปยังโฮสท์ของตัวเอง เพื่อทำการแจ้งให้ทราบว่า การเชื่อมต่อสำเร็จแล้วการทำงานเป็นไปดังรูปที่ 2.17

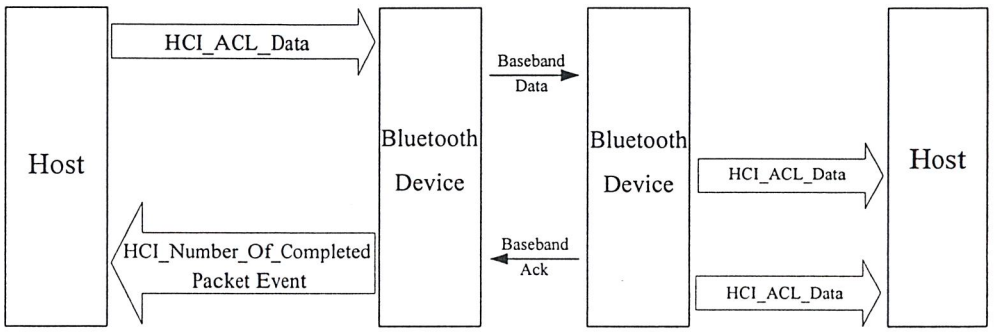


รูปที่ 2.17 การเชื่อมต่อระหว่างอุปกรณ์บลูทูธ

2.13.4 การส่งและการรับข้อมูล

การส่งข้อมูล จะแบ่งออกเป็น 2 ประเภท คือการส่งข้อมูลแบบ ACL (Asynchronous ConnectionLess) เป็นกรเชื่อมต่อที่ใช้สำหรับการส่งข้อมูลทั่วไป และการส่งข้อมูลแบบ SCO (Synchronous Connection-Oriented) เป็นการเชื่อมต่อที่ใช้สำหรับการส่งข้อมูลที่เป็นเสียงในที่นี้จะขออธิบายการส่งและรับข้อมูลแบบ ACL เท่านั้น

เมื่ออุปกรณ์บลูทูธทำการเชื่อมต่อสำเร็จแล้ว ถ้าอุปกรณ์ต้องการส่งข้อมูลโฮสท์จะทำการส่ง HCI_ACL_Data ไปยังอุปกรณ์บลูทูธจากนั้นอุปกรณ์บลูทูธอีกฝั่งหนึ่ง จะทำการส่ง HCI_ACL_Data ไปยังโฮสท์ ของตัวเองจากนั้นอุปกรณ์บลูทูธที่เป็นตัวส่งสัญญาณ จะทำการส่งจำนวนของข้อมูลการส่งที่สมบูรณ์ ไปยังโฮสท์ของตัวเอง ในรูปของคำสั่ง HCI_Number_Of_Completed Packets Event โดยการทำงานเป็นไปดังรูปที่ 2.18



รูปที่ 2.18 การส่งและการรับข้อมูล

2.14 แนวทางการพัฒนา

(1) คอมพิวเตอร์ไร้สาย

ในปัจจุบัน คอมพิวเตอร์ 1 เครื่อง ต้องเชื่อมต่อกับอุปกรณ์ภายนอกเป็นจำนวนมาก ไม่ว่าจะเป็นพริ้นเตอร์ คีย์บร์อด เมาส์ หรือลำโพง การเชื่อมต่ในปัจจุบันนี้ก็ใช้สายสัญญาณเป็นตัวเชื่อมต่อเกือบทั้งหมด ซึ่งทำให้เกิดความลำบากทั้งในด้านการใช้งาน การเคลื่อนย้าย และความเป็นระเบียบ ถ้าเปลี่ยนการใช้สายไฟเชื่อมต่ออุปกรณ์ต่าง ๆ ทั้งหมดนี้มาเป็น บลูทูธ ก็จะทำให้ปัญหาดังกล่าวหมดไปทันที อีกทั้งยังง่ายต่อตัวผู้ใช้เพราะการเชื่อมต่อทั้งหมดจะดำเนินไปโดยอัตโนมัติ ไม่ต้องกังวลเรื่องการต่อสายผิดตำแหน่งอีกต่อไป

(2) ชุดหูฟัง

ในปัจจุบัน การแก้ปัญหาของผู้ใชโทรศัพท์ที่ต้องการใช้มือทั้งสองข้างทำงานอย่างอื่นไปพร้อม ๆ กันด้วย คือการใช้ชุดหูฟัง โดยชุดหูฟังดังกล่าวจะมีสายเชื่อมต่อจากตัวโทรศัพท์มายังหูฟังที่ติดอยู่กับตัวผู้ใช้

นั่นหมายความว่า ตัวผู้ใช้ไม่สามารถเคลื่อนตัวไปได้ไกลกว่าที่สายจะยาวถึง แต่เมื่อทดแทนสายดังกล่าวนี้ด้วย บลูทูธ ผู้ใช้สามารถขยับตัวไปไหนได้อย่างสะดวกไม่ต้องคอยระวังเรื่องสายอีกต่อไป ถ้าในกรณี สมอลทอร์ก ของโทรศัพท์มือถือ ผู้ใช้สามารถเก็บโทรศัพท์ไว้ที่ไหนก็ได้ไกล้ ๆ ตัวในขณะที่ใช้ชุดหูฟังบลูทูธ และเนื่องจากชุดหูฟัง บลูทูธ ไม่ได้เชื่อมต่อตายตัวกับอุปกรณ์ใด ดังนั้น จึงเป็นไปได้ว่าหูฟังตัวเดียวกันนี้ เมื่อไม่ได้ใช้งานเป็น สมอลทอร์ก ก็สามารถฟังเพลงจากเครื่องเล่นซีดี หรือเป็นชุดหูฟังของโทรศัพท์บ้านอีกเครื่องก็ได้

(3) อินเทอร์เน็ต บริดจ์

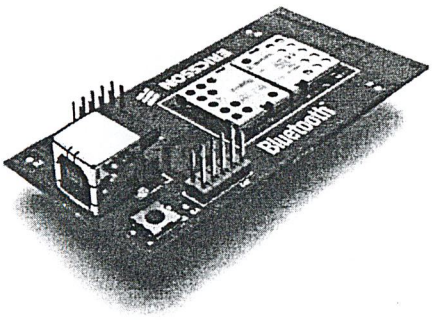
ในปัจจุบัน หากต้องการเชื่อมต่ออุปกรณ์พกพาต่าง ๆ ไม่ว่าจะเป็นคอมพิวเตอร์โน้ตบุ๊ค หรือพ็อกเก็ตพีซี เข้ากับอินเทอร์เน็ต จำเป็นต้องเชื่อมต่ออุปกรณ์ดังกล่าวเข้ากับช่องทางสื่อสาร ซึ่งอาจจะเป็นโทรศัพท์มือถือ หรือสายโทรศัพท์ธรรมดาผ่านทางสายเชื่อมต่อ แต่การเชื่อมต่อดังกล่าวสามารถใช้งานได้เพียง 1 อุปกรณ์ ต่อ 1 ครั้ง และยังเกิดปัญหาในเรื่องความเกะกะของสายเมื่อต้องใช้งานนอกสถานที่ หรือในยานพาหนะต่าง ๆ แต่เมื่อทดแทนด้วย บลูทูธ เครื่องคอมพิวเตอร์ หรืออุปกรณ์พกพาต่าง ๆ จะสามารถเชื่อมต่อเข้ายังอินเทอร์เน็ตได้ โดยต่อผ่านโทรศัพท์มือถือที่มีระบบ GPRS โดยไม่จำเป็นต้องใช้สาย ซึ่งจะช่วยลดความยุ่งยากลง อีกทั้งยังเพิ่มความสะดวกสบายในการทำงานขึ้นด้วย

บทที่3

การทดลอง

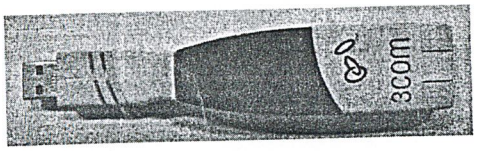
3.1 โมดูลบลูทูธ

โครงงานนี้เลือกใช้ ชุดทดลองการสื่อสารไร้สาย บลูทูธแอพลิเคชั่นทูลคิท (Bluetooth Application Tool Kit) ของบริษัท คอมเทค เทเลก้า (Comtec Teleca) ซึ่งใช้โมดูล บลูทูธ rok 101007 เป็นส่วนประกอบ ทำงาน แบบ point-to-multipoint ดังแสดงในรูปที่ 3.1 โดยนำบลูทูธแอพลิเคชั่นทูลคิท มาทำการเชื่อมต่อเข้ากับบอร์ดไมโครคอนโทรลเลอร์ ผ่านทางซีเรียลพอร์ต



รูปที่ 3.1 Bluetooth Application Tool Kit (BD_ADDR=0x00803715C7CF)

และอุปกรณ์บลูทูธ อีกตัวที่นำมาทำการทดลองก็คือ อุปกรณ์บลูทูธ ของบริษัท 3 Com ใช้เชื่อมต่อกับคอมพิวเตอร์ ดังรูปที่ 3.2



รูปที่ 3.2 อุปกรณ์บลูทูธ ของบริษัท 3 Com (BD_ADDR=0x000476E1B669)

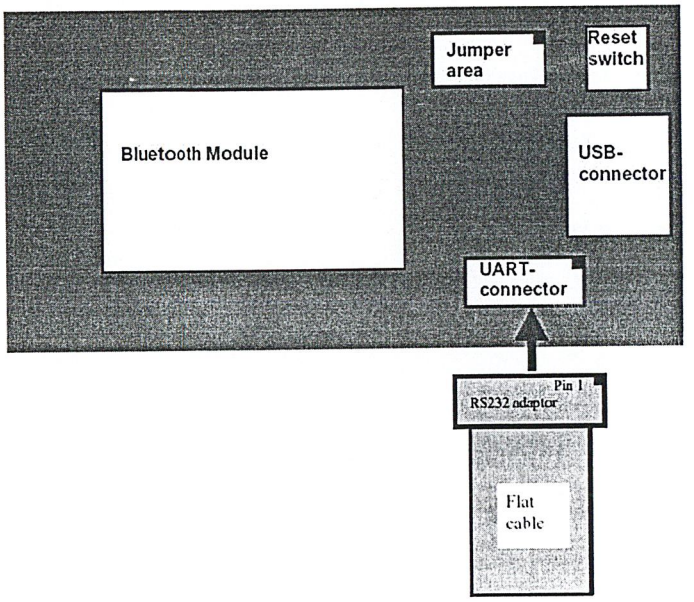
3.1.1 ข้อมูลทั่วไปของ โมดูลบลูทูธ rok 101007

- มีกำลังส่งอยู่ในคลาสที่ 2 , 0 dBm และระยะการใช้งานประมาณ 10 เมตร
- บลูทูธ เวอร์ชั่น 1.0b
- FCC และ ETSI ได้รับการยอมรับสำหรับข้อกำหนดของคลื่นความถี่วิทยุ
- อัตราการส่งข้อมูลสูงสุดของ UART คือ 460 kb/s
- มีพอร์ตการเชื่อมต่อให้เลือกใช้ 3 พอร์ตด้วยกัน คือ
  - พอร์ต UART สำหรับการส่งข้อมูล (data)

- พอร์ต PCM สำหรับการส่งข้อมูลเสียง (voice)
- พอร์ต USB สำหรับการส่งข้อมูล (data)

- เลเยอร์ระดับล่าง ๆ ตั้งแต่ HCI ลงมาถึง Radio ถูกรวมไว้ในโมดูลตัวนี้ด้วย
- มีสายอากาศในตัว
- ทำงานแบบ point-to-multipoint

3.1.2 ส่วนประกอบทางฮาร์ดแวร์ของ บลูทูธแอพพลิเคชั่นทูลคิท นี้จะประกอบไปด้วย บลูทูธโมดูล rok 101007 , Jumper area (พอร์ต PCM) , ส่วนเชื่อมต่อแบบ UART , ส่วนเชื่อมต่อแบบ USB และปุ่มรีเซต ดังแสดงในรูปที่ 3.3 และตารางที่ 3.1 , 3.2 และ 3.3 เป็นการบอกว่าทุกขาของแต่ละพอร์ตทำงานอย่างไร



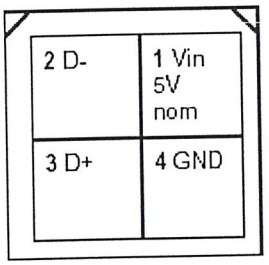
รูปที่ 3.3 ส่วนประกอบทางฮาร์ดแวร์ของ บลูทูธแอพพลิเคชั่นทูลคิท

9 RESET (R3)	**7** PCM_IN (A1)	**5** PCM_OUT (A2)	**3** WAKE_UP (B4)	**1** Vin 5V nom
10 GND	**8** DETACH (C1)	**6** VCC 3.3V (C2,C4 and C6)	**4** PCM_SYNK (A3)	**2** PCM_CLK (A4)

ตารางที่ 3.1 การใช้งานแต่ละขาในบอร์ด Jumper area

9	GND	**7**	not used	**5**	TXD (B5)	**3**	RXD (A4)	**1**	not used
10	not used	**8**	not used	**6**	RST (A6)	**4**	CTS (B6)	**2**	not used

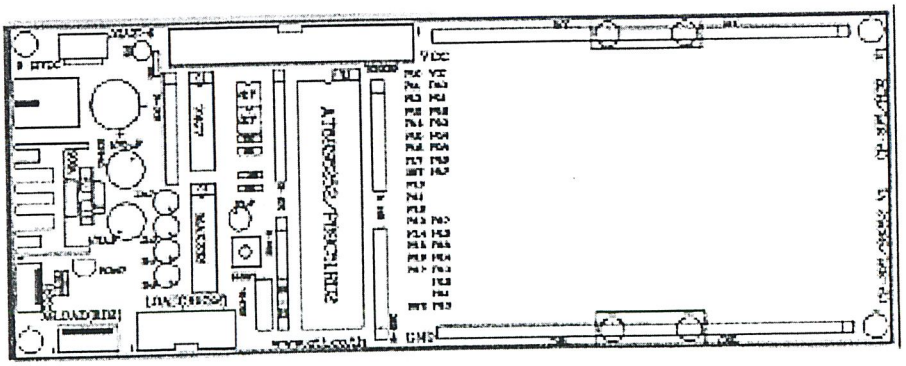
ตารางที่ 3.2 การใช้งานแต่ละขาในบอร์ด UART



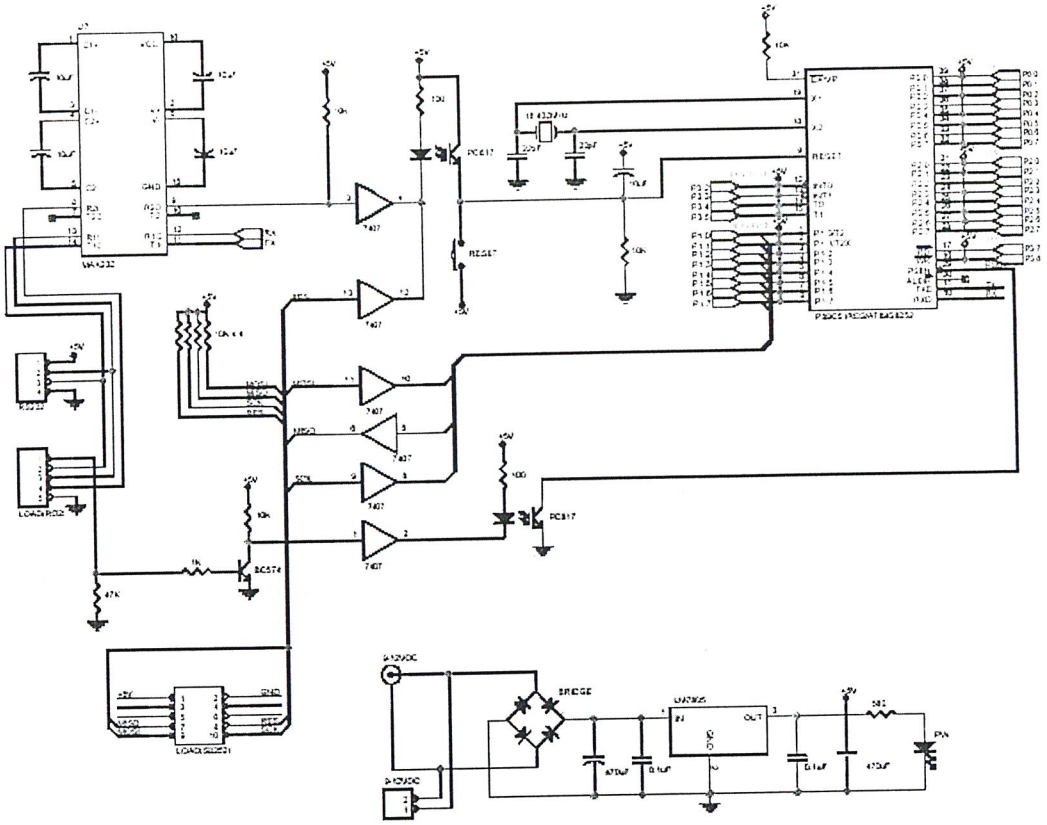
ตารางที่ 3.3 การใช้งานแต่ละขาในบอร์ด USB

3.2 บอร์ดไมโครคอนโทรลเลอร์

โครงงานนี้เลือกใช้บอร์ดไมโครคอนโทรลเลอร์ CP-SPI/RD2 เวอร์ชัน 1.0 ของบริษัท อีทีที (ETT) เหตุที่เลือกใช้งานบอร์ดไมโครคอนโทรลเลอร์นี้ ก็เนื่องจากบอร์ดนี้ใช้ ซีพียู(CPU) P89C51RD2 ที่สนับสนุนอัตราการส่งข้อมูลความเร็วสูง และในการทดลองนี้ใช้ไมโครคอนโทรลเลอร์เชื่อมต่อกับอุปกรณ์บลูทูธ ซึ่งอุปกรณ์บลูทูธนี้ต้องการ อัตราการส่งข้อมูลถึง 57,600 kb/s ทั้งนี้ยังสามารถดาวโหลดโปรแกรมจากคอมพิวเตอร์ลงในบอร์ด โดยผ่านทาง RS232 โดยไม่ต้องมีอุปกรณ์อื่นมาเสริม รูปที่ 3.4 แสดงตำแหน่งการวางอุปกรณ์ และรูปที่ 3.5 แสดงโครงสร้างภายใน ของบอร์ด CP-SPI/RD2 เวอร์ชัน 1.0

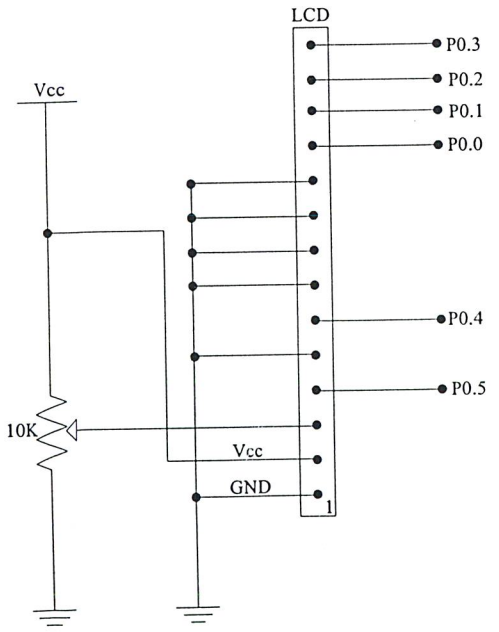


รูปที่ 3.4 ตำแหน่งการวางอุปกรณ์ ของบอร์ด CP-SPI/RD2 เวอร์ชั่น 1.0



รูปที่ 3.5 โครงสร้างภายใน ของบอร์ด CP-SPI/RD2 เวอร์ชั่น 1.0

ทั้งนี้ ได้ทำการต่อจอแอลซีดี (LCD) กับบอร์ดไมโครคอนโทรลเลอร์ เพื่อทำการแสดงผลการทดลอง ดังรูปที่ 3.6



รูปที่ 3.6 วงจร แอลซีดี

3.3 การทดลอง

3.3.1 ศึกษาการใช้งานอุปกรณ์บลูทูธ

3.3.2 ใช้โปรแกรม ซีเรียลมอนิเตอร์ (Serial Monitor) จับข้อมูลที่วิ่งผ่านพอร์ต UART ของอุปกรณ์บลูทูธ โดยนำ อุปกรณ์บลูทูธ (Bluetooth application tool kit) ทั้งสองต่อเข้ากับคอมพิวเตอร์ แล้วรันโปรแกรมแอพพลิเคชันที่มากับตัวอุปกรณ์

3.3.3 นำข้อมูลที่ได้จากโปรแกรม ซีเรียลมอนิเตอร์ มาวิเคราะห์ลำดับการทำงาน ทำให้ทราบว่าอุปกรณ์บลูทูธที่ใช้ สามารถทำงานได้โดยรับคำสั่งจากอุปกรณ์ควบคุมเป็นแพ็กเกจ โดยในแต่ละแพ็กเกจสามารถจำแนกได้จากเฮดเดอร์ (Header) ของแต่ละแพ็กเกจ ดังตารางที่ 3.4

HCI packet type	HCI packet indicator
HCI Command Packet	0x01
HCI ACL Data Packet	0x02
HCI SCO Data Packet	0x03
HCI Event Packet	0x04
Error Message Packet	0x05
Negotiation Packet	0x06

ตารางที่ 3.4 แสดงประเภทของคำสั่ง (HCI packet type)

- HCI Command Packet เป็นแพ็กเกจที่อุปกรณ์ควบคุม (Host) ใช้ส่งคำสั่งควบคุมไปให้อุปกรณ์ บลูทูธ (Host Controller) ซึ่งจะรับคำสั่งไปปฏิบัติตามต่อไป
- HCI ACL Data Packet เป็นแพ็กเกจสำหรับ ใช้ส่งข้อมูลแบบอะซิงโครนัส ระหว่างอุปกรณ์ควบคุมกับอุปกรณ์บลูทูธ (Asynchronous)
- HCI SCO Data Packet เป็นแพ็กเกจสำหรับ ใช้ส่งข้อมูลแบบซิงโครนัส ระหว่างอุปกรณ์ควบคุมกับอุปกรณ์บลูทูธ (Synchronous)
- HCI Event Packet เป็นแพ็กเกจที่ อุปกรณ์บลูทูธตอบกลับมาที่ อุปกรณ์ควบคุม หลังจากที่อุปกรณ์ควบคุม ส่งคำสั่งไปให้อุปกรณ์ บลูทูธ ทุกครั้ง
- Error Message Packet เป็นแพ็กเกจที่ อุปกรณ์บลูทูธ ตอบกลับมาที่ อุปกรณ์ควบคุม เพื่อรายงานความผิดพลาดของแพ็กเกจที่ส่งรับกัน
- Negotiation Packet เป็นแพ็กเกจที่ใช้สำหรับตั้งค่าการใช้งานเบื้องต้นระหว่าง อุปกรณ์ควบคุม กับ อุปกรณ์บลูทูธ

การทดลองนี้พิจารณาเฉพาะ HCI Command Packet , HCI ACL Data Packet และ HCI Event Packet เมื่อทราบชนิดของแพ็กเกจแล้ว นำส่วนอื่น ๆ ของแพ็กเกจนั้น มาพิจารณาดังตัวอย่างต่อไปนี้

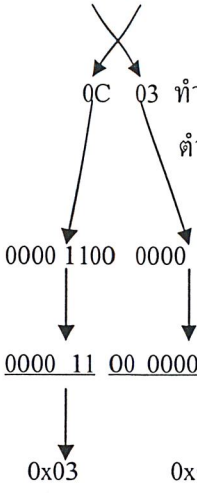
ตัวอย่าง คำสั่งที่ 1 จาก ซีเรียลมอนิเตอร์

Request : 01 03 0C 00

Answer : 04 0E 04 08 03 0C 00

อธิบาย จาก Request 01 คือ HCI Command Packet

03 0C คือ OpCode สามารถแปลงเพื่อดูชนิดของคำสั่งได้ดังต่อไปนี้



0C 03 ทำการสลับตำแหน่งเนื่องจากโปรแกรม Serial Monitor จะสลับตำแหน่งกับอยู่

0000 1100 0000 0011 แปลงให้เป็นเลขฐานสอง

0000 11 00 0000 0011 แปลงเป็น OGF 6 บิต OCF 10 บิต

0x03 0x0003 ได้ OGF และ OCF ซึ่งเป็นเลขฐาน 16

จะได้ว่า OGF มีค่าเป็น 0x03 เป็นคำสั่งที่อยู่ในชุด Host Controller & Baseband Command และ OCF มีค่าเป็น 0x0003 ซึ่งตรงกับคำสั่งรีเซตในชุดนี้ (สามารถดูได้จากดาต้าชีทของ Bluetooth Application Tool Kit) ส่วนค่าที่อยู่ต่อจาก OpCode เป็นค่าพารามิเตอร์

จาก Answer 04 คือ HCI Event Packet

0E 04 08 เป็นค่าพารามิเตอร์

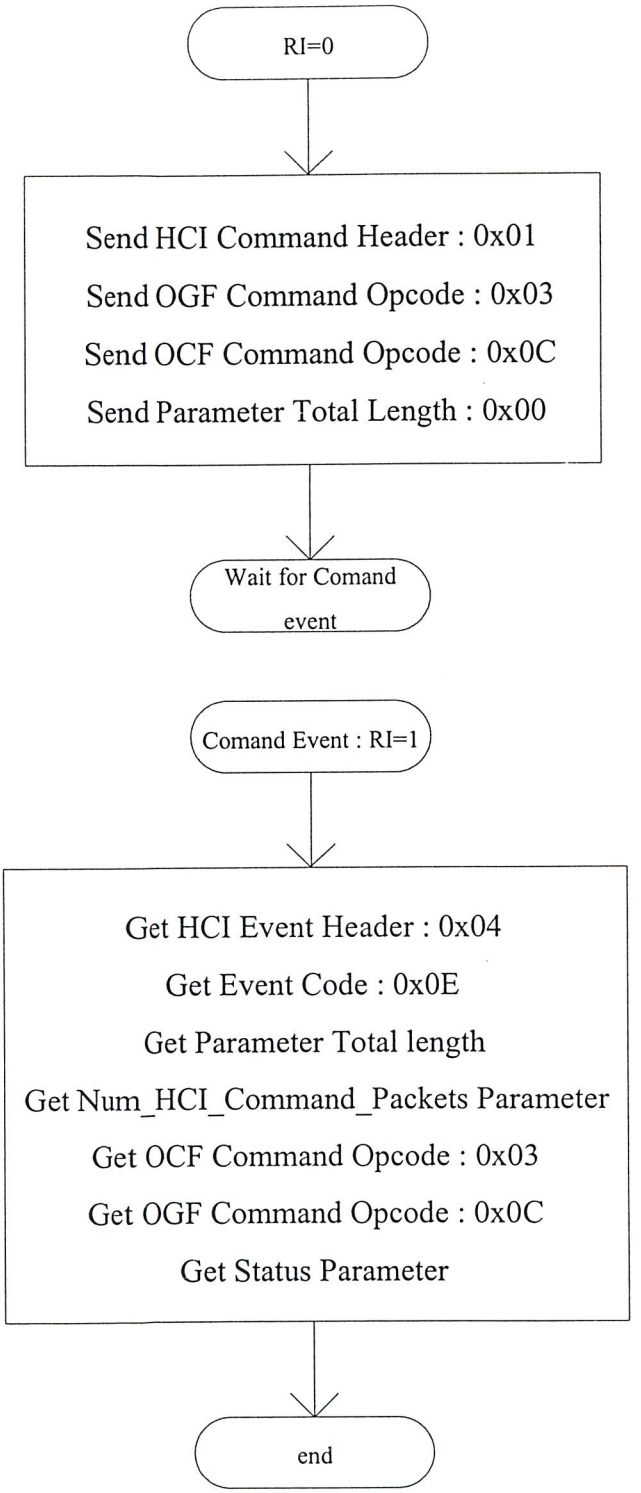
03 0C เป็นค่า OpCode ที่ตอบกลับมาซึ่งต้องมีค่าตรงกับ Request

00 เป็นค่าที่บอกว่ามีการ error หรือไม่ถ้าไม่มีจะตอบกลับมาเป็น 00 แต่ถ้าตอบกลับมาเป็นค่าอื่น ที่ไม่ใช่ 00 สามารถดูได้ว่าเกิด error เพราะอะไร จากดาต้าชีท

ทำเช่นเดียวกัน กับทุก ๆ แพ็กเกจ ทำให้สามารถเขียนโฟลว์ชาร์ตเพื่อนำไปเขียนไมโครคอนโทรลเลอร์ได้ดังนี้

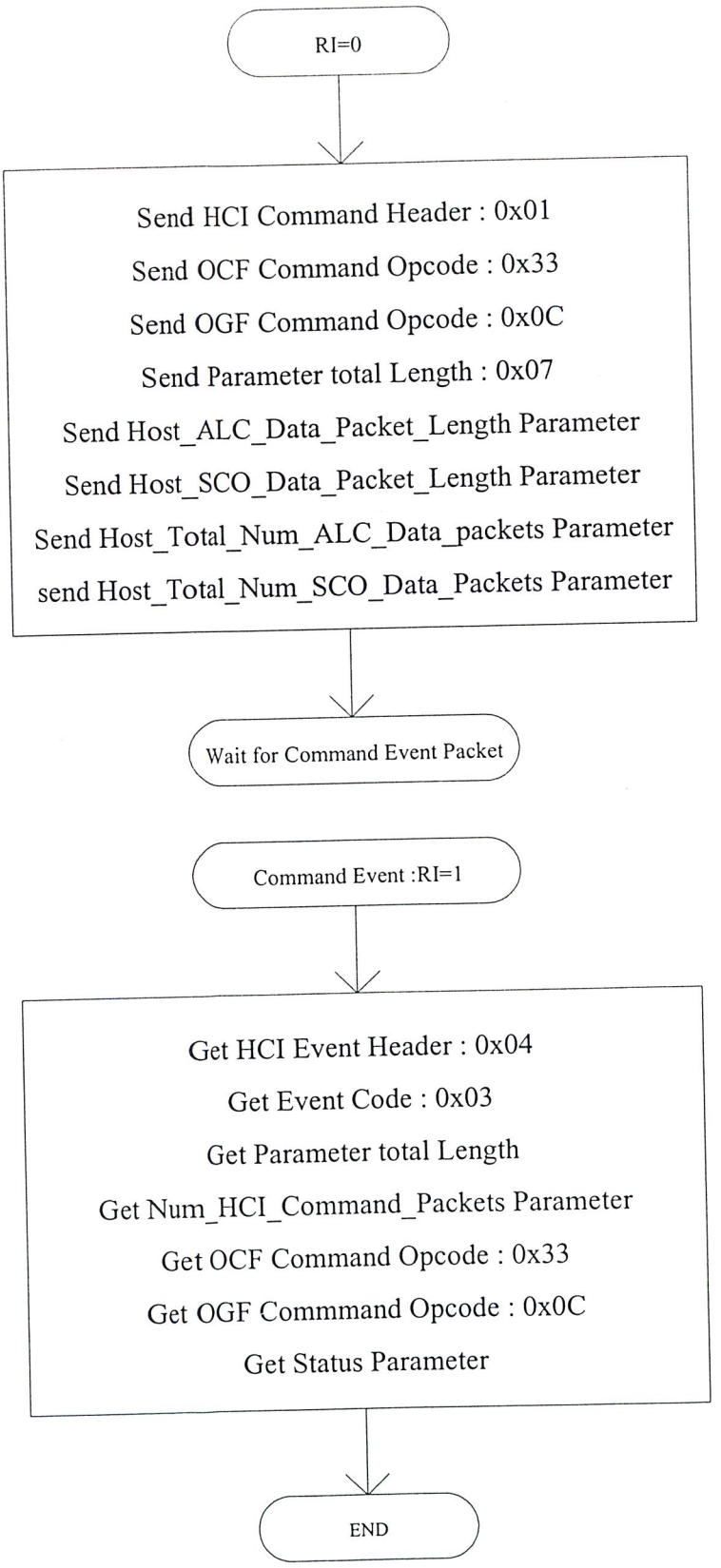
1. คำสั่งที่ใช้ในการรีเซต (Reset) :OpCode = 0x0C03

คำสั่งรีเซตจะการรีเซต Host Controller และ Link Manager ทำการรีเซตที่ตัวอุปกรณ์เท่านั้น หลังจากทำการรีเซตแล้ว อุปกรณ์ บลูทูธพร้อมที่จะทำงาน Host Controller จะทำการส่งคำสั่งตอบกลับมาโดยอัตโนมัติ ซึ่งส่งกลับมาเป็น Event Packet



2. คำสั่งที่ใช้ในการกำหนดขนาดของ Host (Host_Buffer_Size): OpCode = 0x0C33

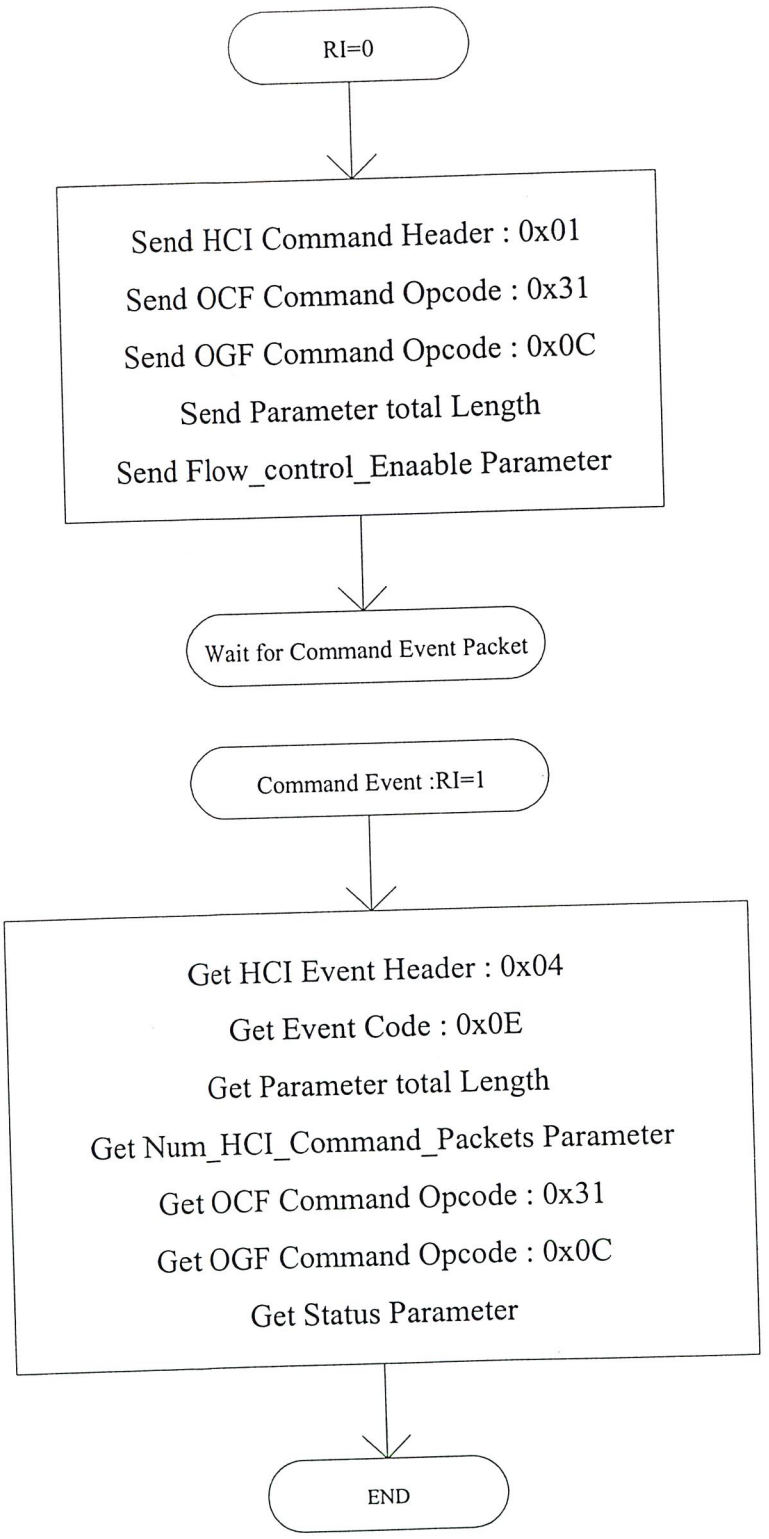
คำสั่ง Host_Buffer_Size ถูกใช้โดย Host เพื่อบอกขนาดแพ็กเกจของ HCI ALC และ SCO จาก Host Controller ถึง Host



3. **คำสั่งที่ใช้ในการควบคุมจาก Host ถึง Host Controller**

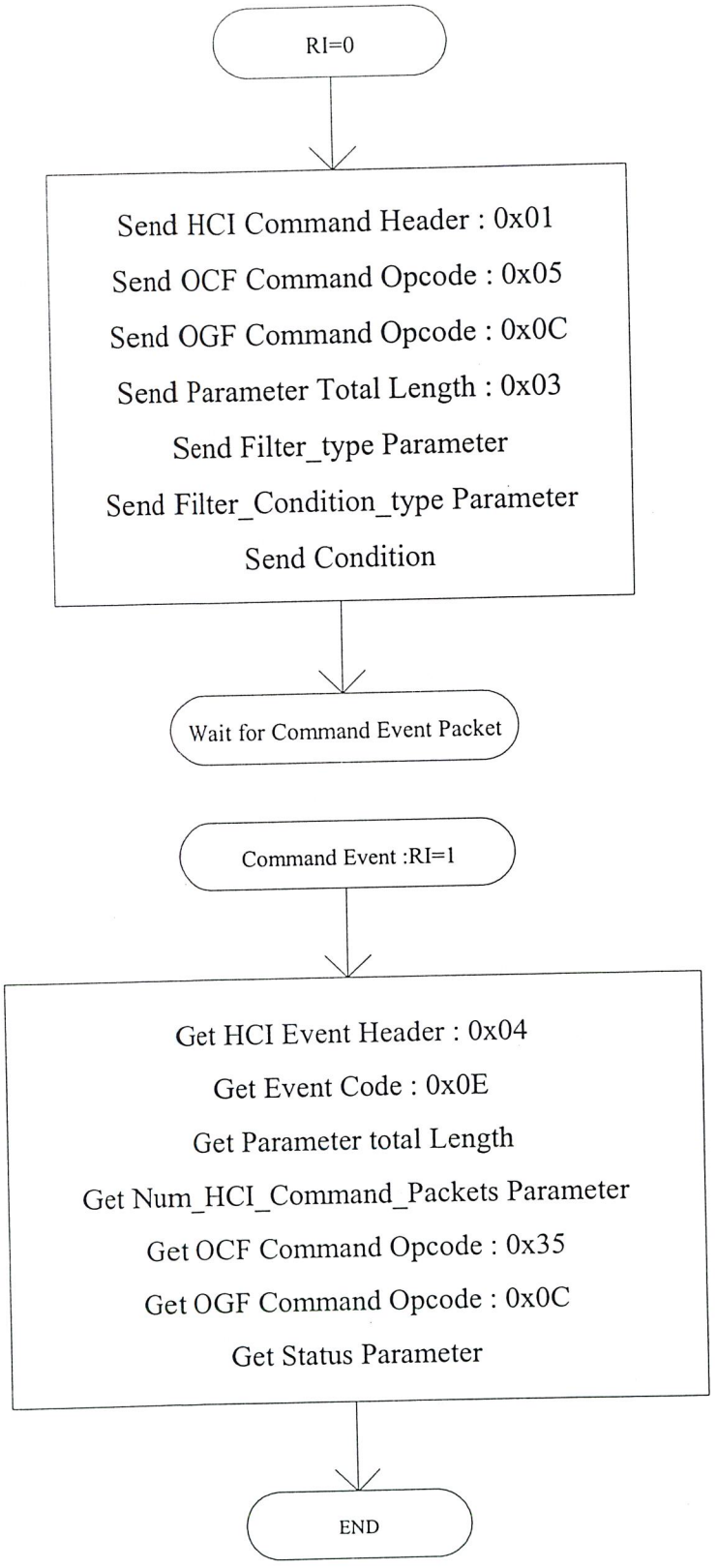
(Set_Host_Controller_To_Host_Flow_Control) : OpCode = 0x0C31

คำสั่ง Set_Host_Controller_To_Host_Flow_Control ใช้เปิดปิด Flow Control สำหรับการส่งสัญญาณเสียงหรือข้อมูลจาก Host Controller ถึง Host



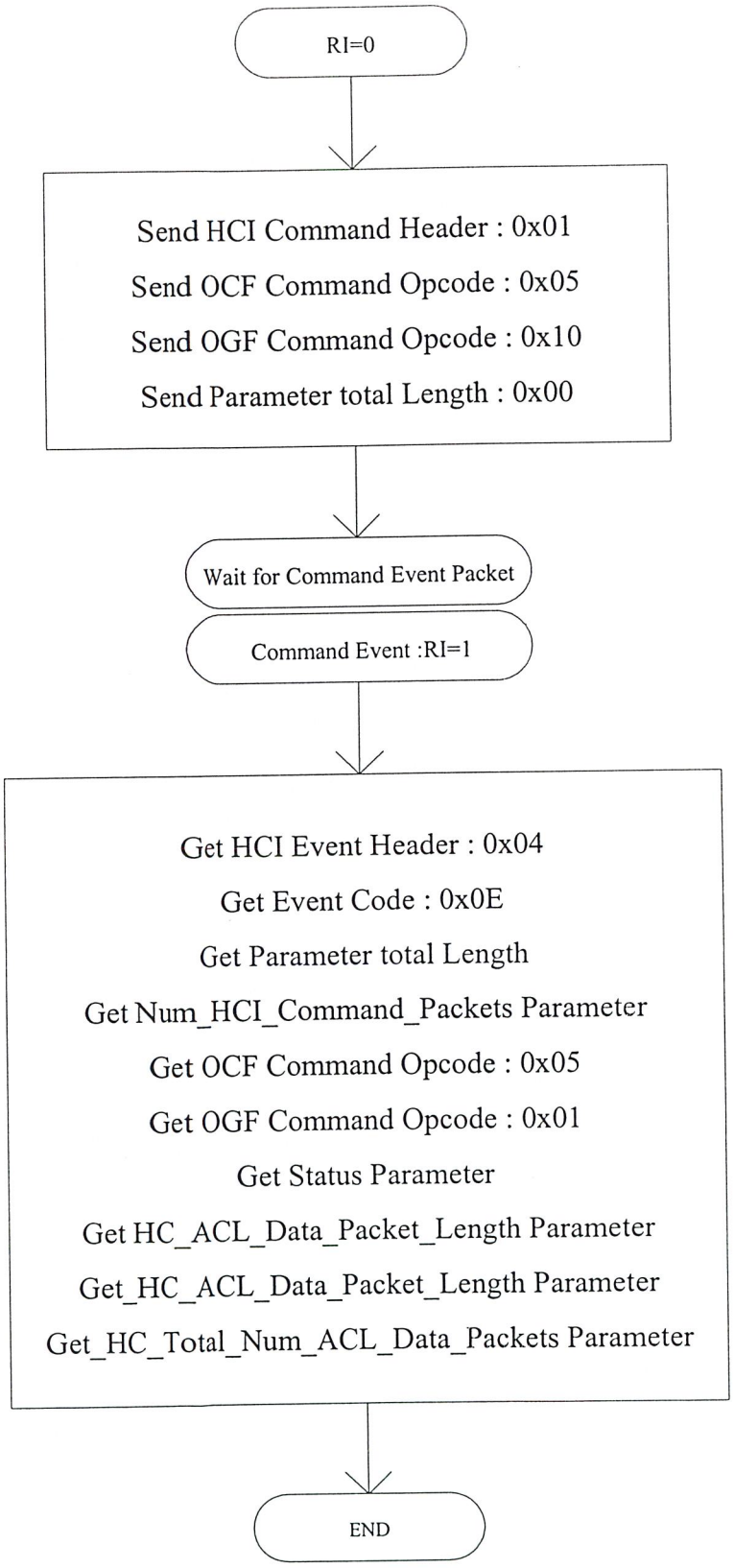
4. คำสั่งที่ใช้ในการเซตค่า **Filter (Set_Evetn_Filter):OpCode=0x0C05**

เป็นคำสั่งที่ Host ใช้ระบุความแตกต่างของเหตุการณ์ต่าง ๆ



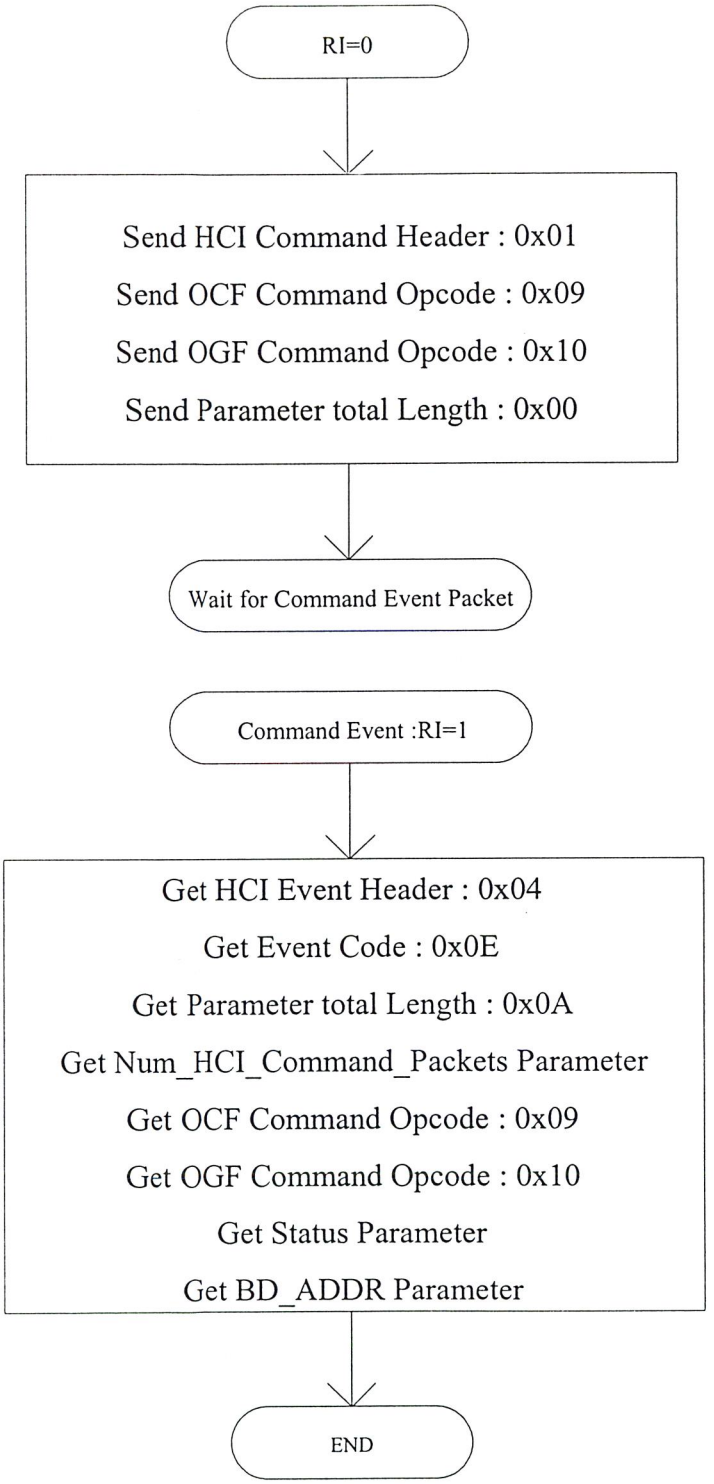
5. คำสั่งที่ใช้ในการอ่านค่า **Buffer (Read_Buffer_Size) :OpCode=0x1005**

เป็นคำสั่งที่ถูกใช้ในการอ่านขนาดมากที่สุดส่วนของข้อมูลใน HCL ACL และ SCO data Packet ที่ถูกส่งจาก Host ถึง Host Controller



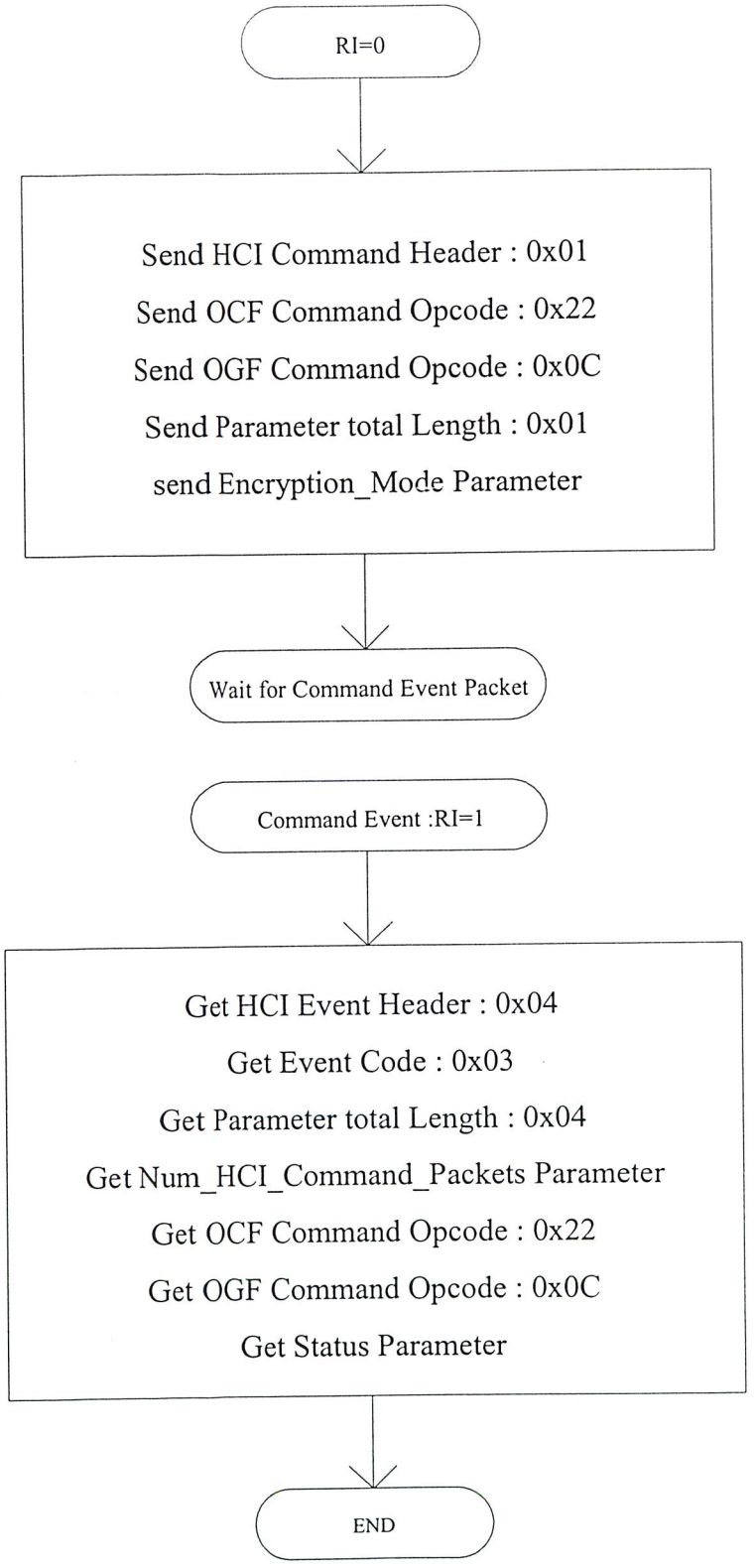
6. คำสั่งที่ใช้ในการอ่านค่าแอดเดรส **(Read_BD_ADDR) :OpCode=0x1009**

เป็นคำสั่งที่ใช้ในการอ่านค่า BD Address ของอุปกรณ์ที่เชื่อมต่อกับ Host อยู่

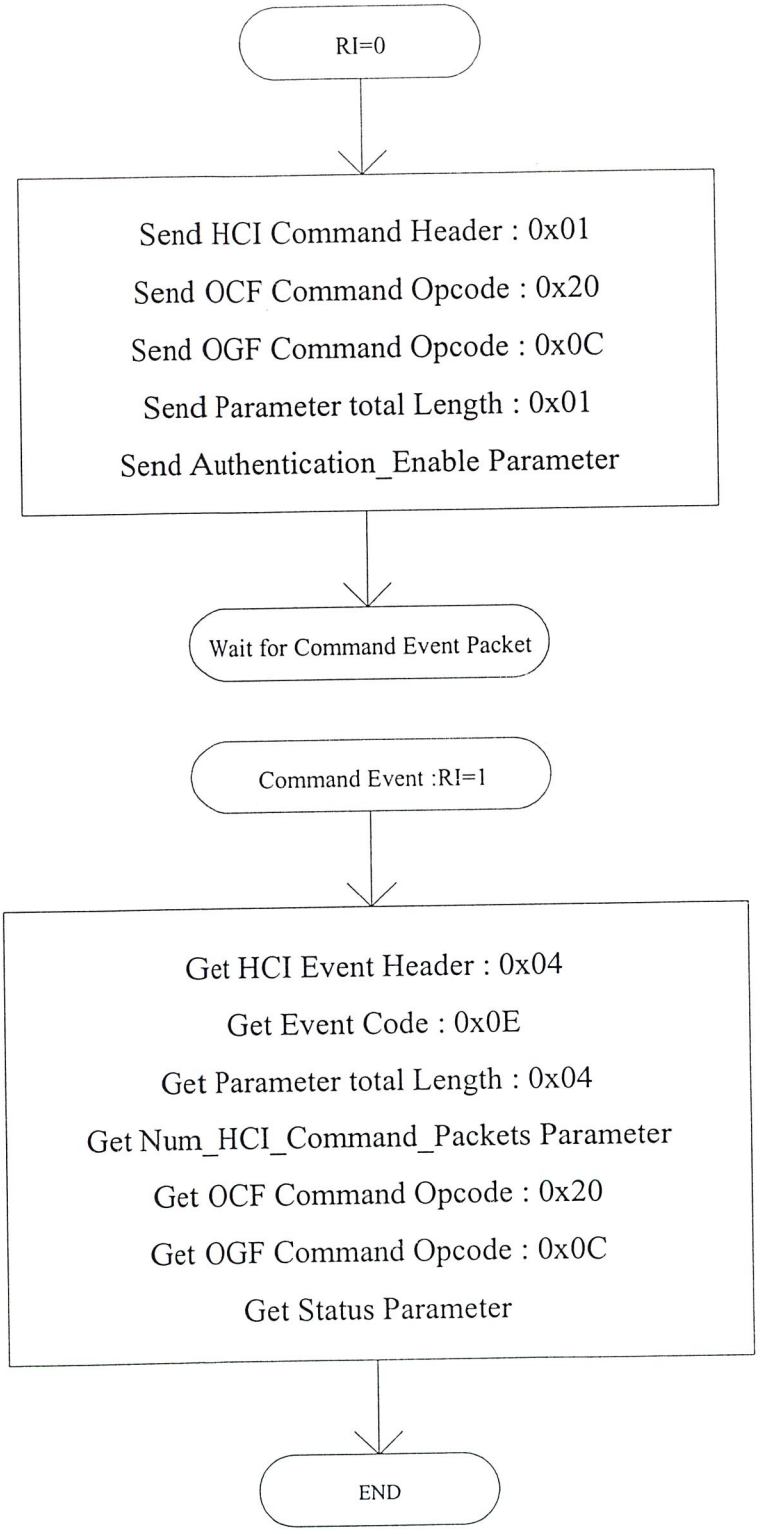


7. คำสั่งที่ในการเข้ารหัส (Write_Encryption_Mode) :OpCode=0x0C22

เป็นคำสั่งที่ใช้ในการเขียนค่าการเข้ารหัสพารามิเตอร์ซึ่งใช้ควบคุมสภาพของอุปกรณ์บลูทูธที่จะเข้ารหัสการติดต่อ ถ้าอุปกรณ์บลูทูธตัวอื่น ๆ ร้องขอ



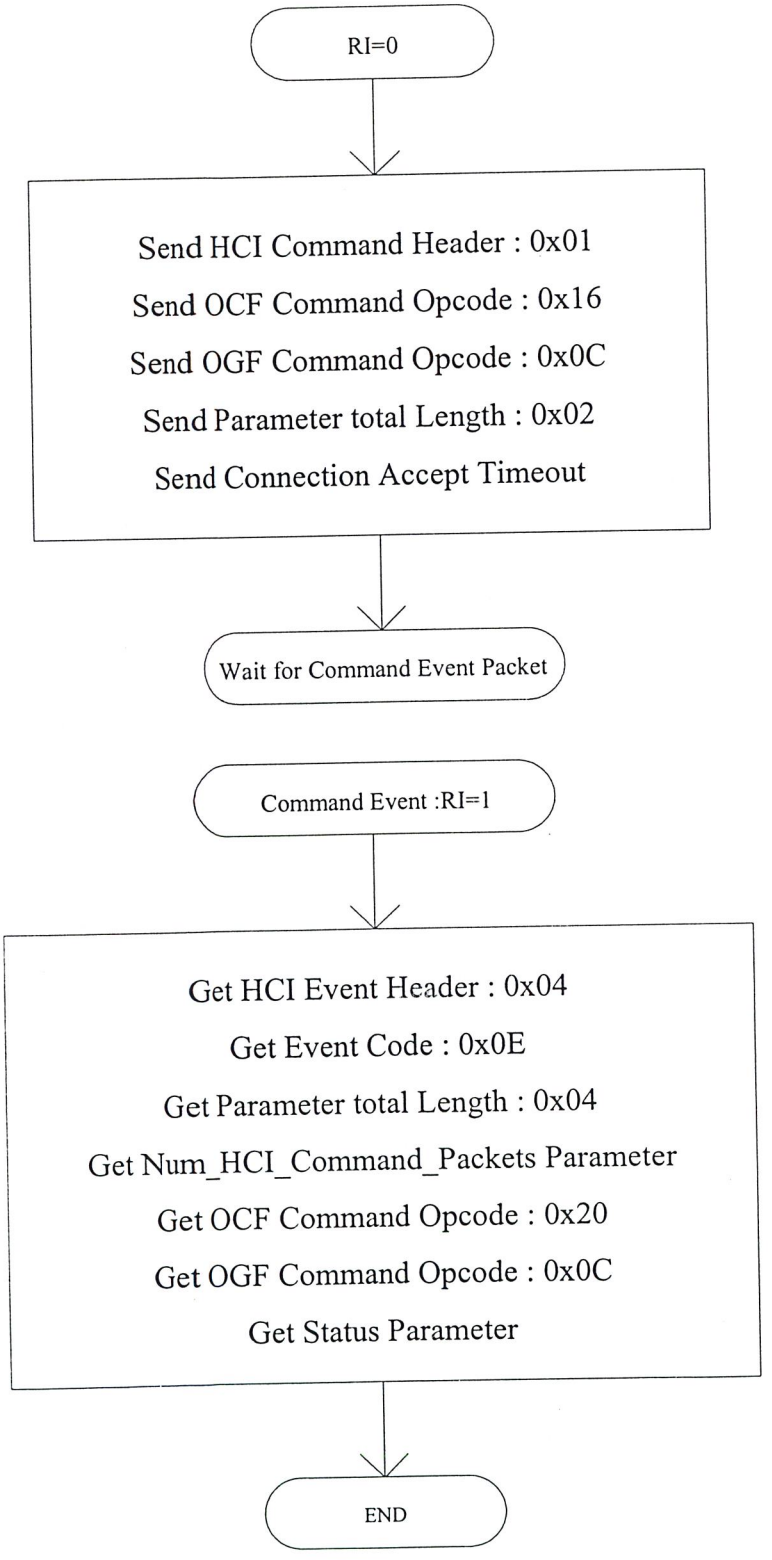
8. คำสั่งที่ใช้ในการยืนยันการทำงาน (Write_Authentication_Enable) : OpCode=0x0C20
เป็นค่าพารามิเตอร์ที่ใช้ควบคุมถ้าอุปกรณ์ทั่วไปต้องการที่จะติดต่อยังไม่สำเร็จ



9. **คำสั่งที่ใช้ในการเขียนค่าของเวลาในการตอบรับก่อนหมดเวลา**

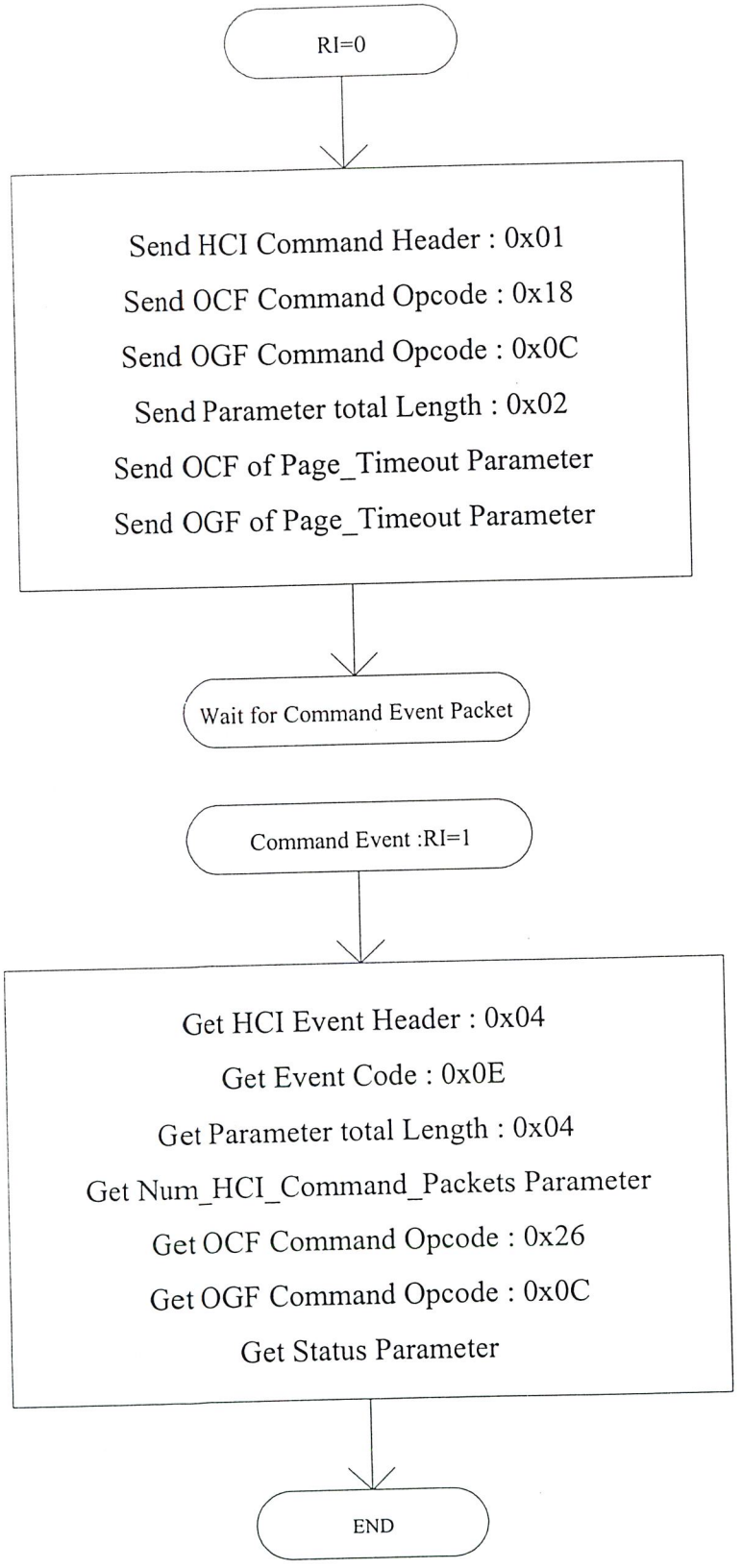
(Write_Connection_Accept_Timeout): OpCode= 0x0C16

เป็นคำสั่งที่ใช้ในการเขียนค่าพารามิเตอร์ซึ่งค่านี้ใช้อนุญาตให้ฮาร์ดแวร์ของอุปกรณ์บลูทูธทำการปฏิบัติการอัตโนมัติในการร้องของการติดต่อ



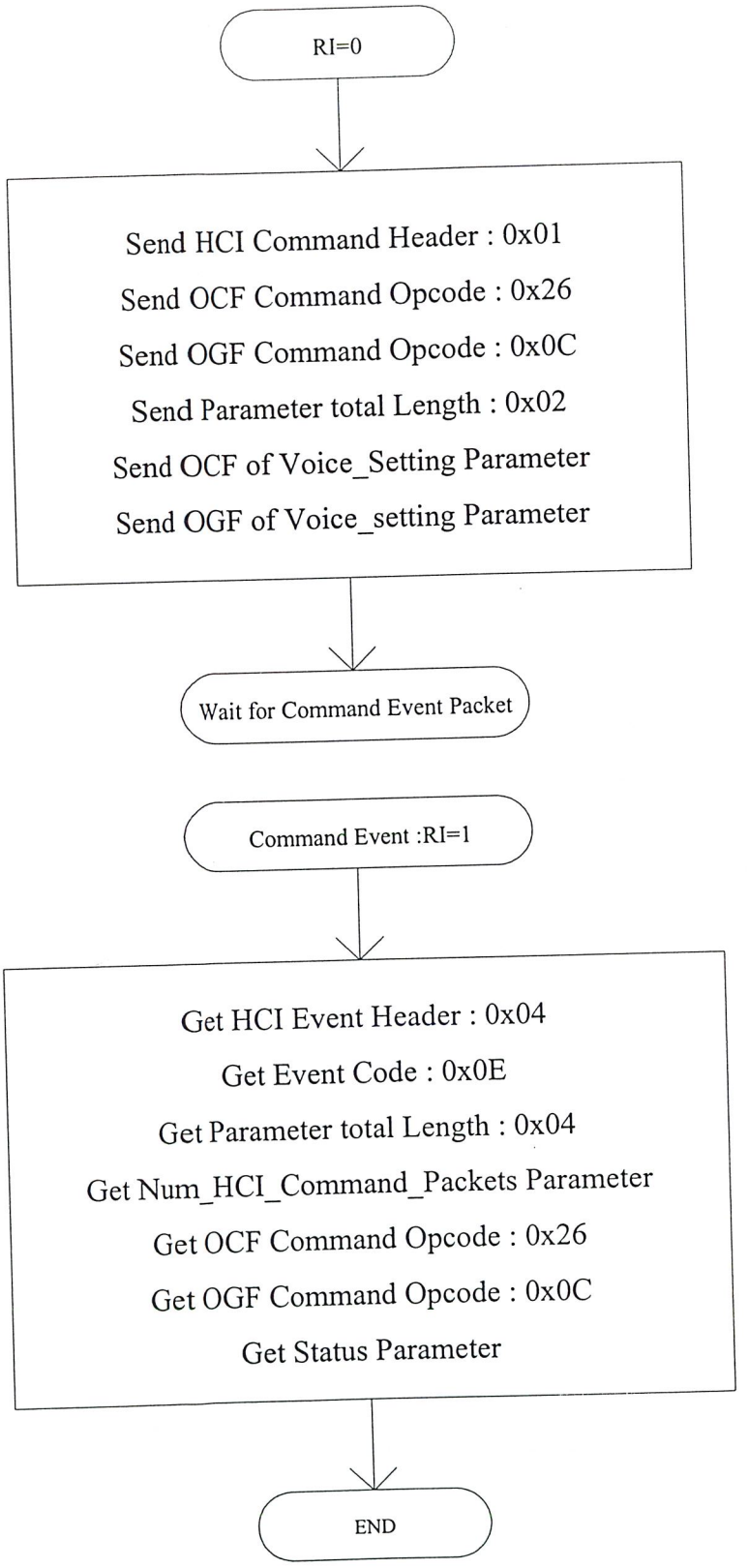
10. คำสั่งที่ใช้ในการเขียนค่าของการเชื่อมต่อระหว่าง Layer Local Link Manager กับ Layer Baseband (Write_Page_Timeout) : OpCode=0x0C18

เป็นคำสั่งที่ใช้เขียนค่าพารามิเตอร์ซึ่งกำหนดเวลาที่มากที่สุดที่ LM รอ Baseband ตอบกลับ



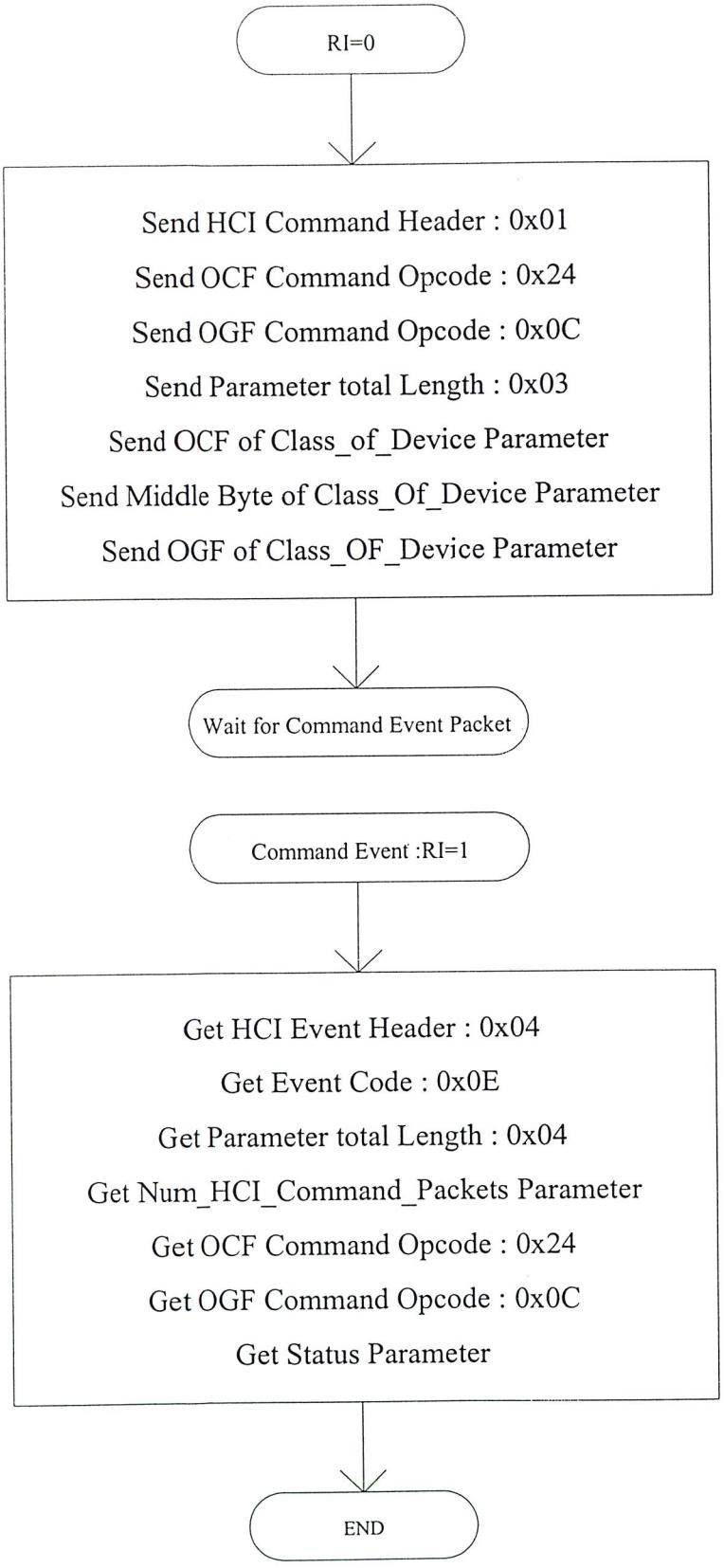
11. คำสั่งที่ใช้ในการเซตค่าของเสียง (Write_Voice_Setting) :OpCode = 0x0C26

ใช้ควบคุมการติดต่อแบบเสียง



12. คำสั่งที่ใช้ เลเยอร์ ต่าง ๆ ของอุปกรณ์ (Write_Class_Of_Device) : OpCode= 0x0C24

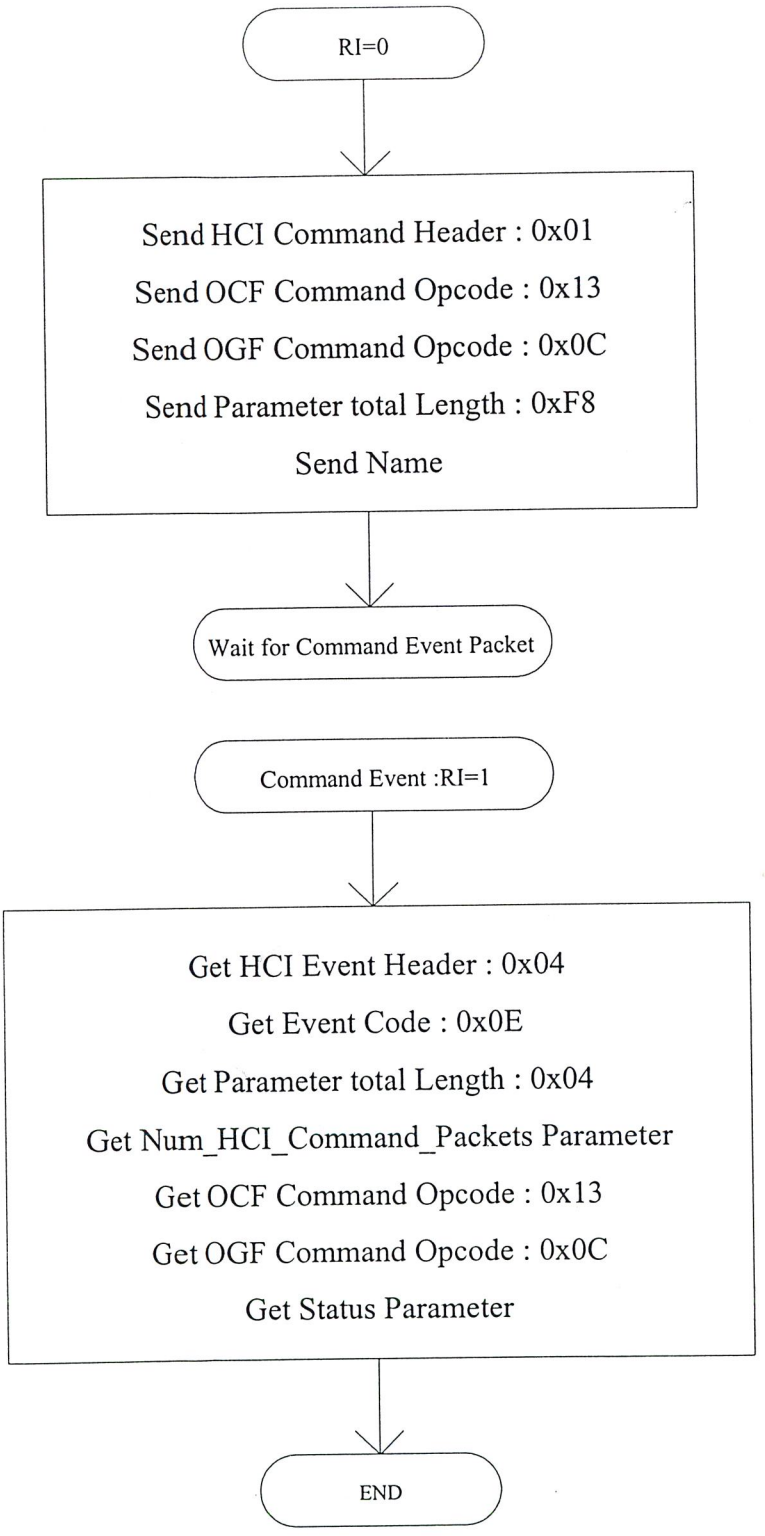
ใช้ชี้ความสามารถของอุปกรณ์อื่น ๆ



13. คำสั่งที่ใช้ในการเปลี่ยนชื่ออุปกรณ์เมื่อทำการติดต่อกันระหว่างอุปกรณ์

(Change_Local_Name) : OpCode = 0x0C13

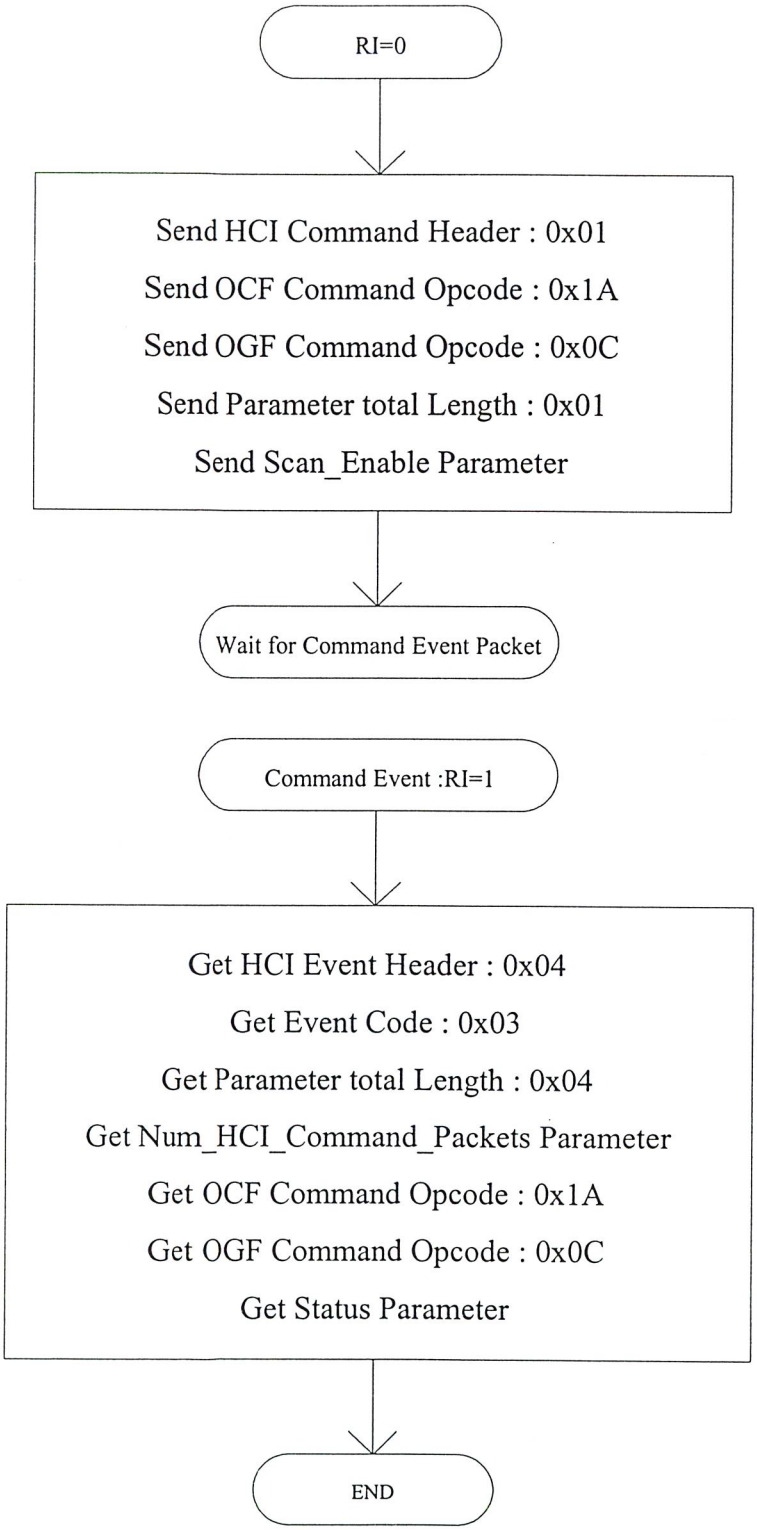
เป็นคำสั่งที่มีไว้สำหรับการเปลี่ยนแปลงชื่ออุปกรณ์บลูทูธ



14. คำสั่งที่ใช้ในการส่งคลื่นสัญญาณของตัวอุปกรณ์

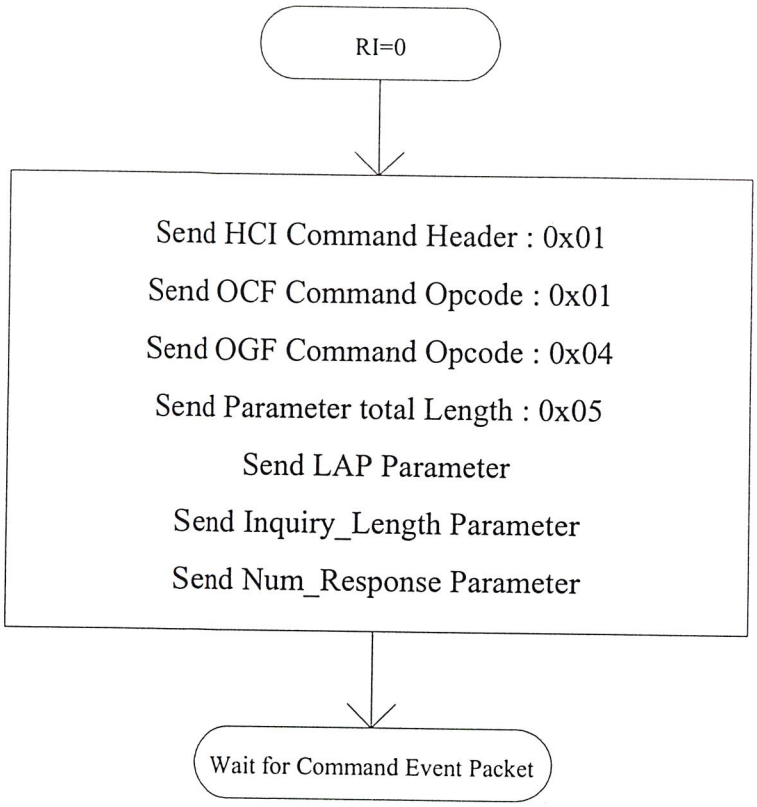
(Write_Scan_Enable) :OpCode = 0x0C1A

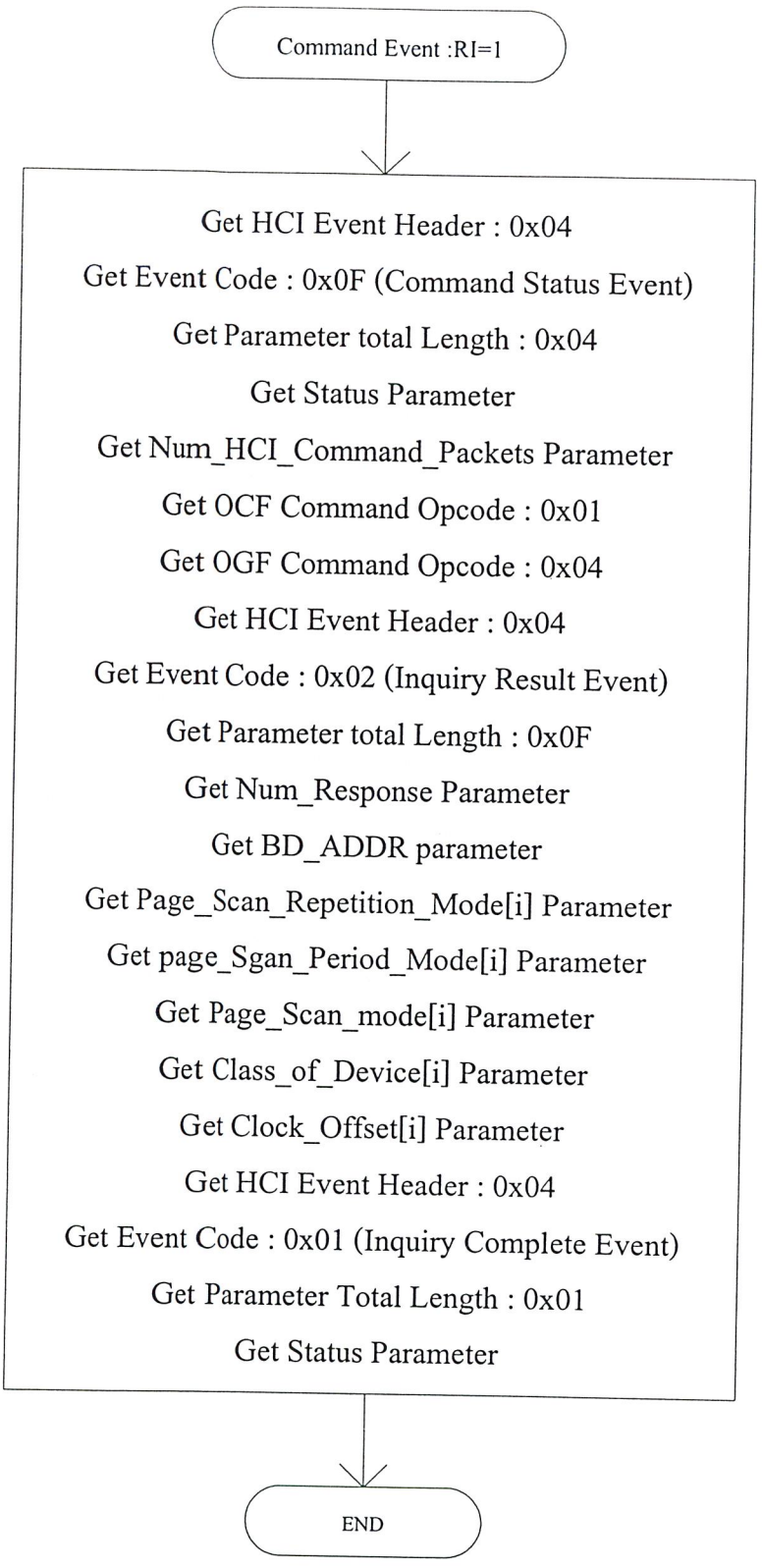
เป็นคำสั่งที่ใช้ในการหาและร้องขอที่จะติดต่อกับอุปกรณ์บลูทูธตัวอื่น ๆ



15. คำสั่งที่ใช้ในการร้องขอ (Inquiry) :OpCode = 0x0401

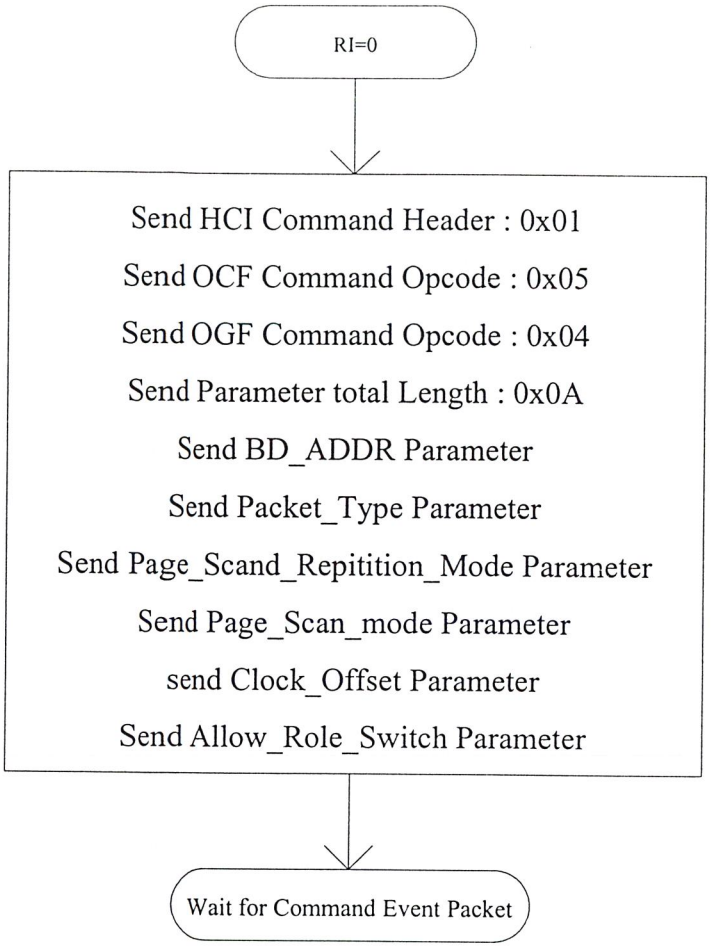
คำสั่งนี้ทำให้อุปกรณ์บลูทูธเข้าสู่โหมดการร้องขอเพื่อนำไปใช้ค้นหาอุปกรณ์บลูทูธรอบข้าง

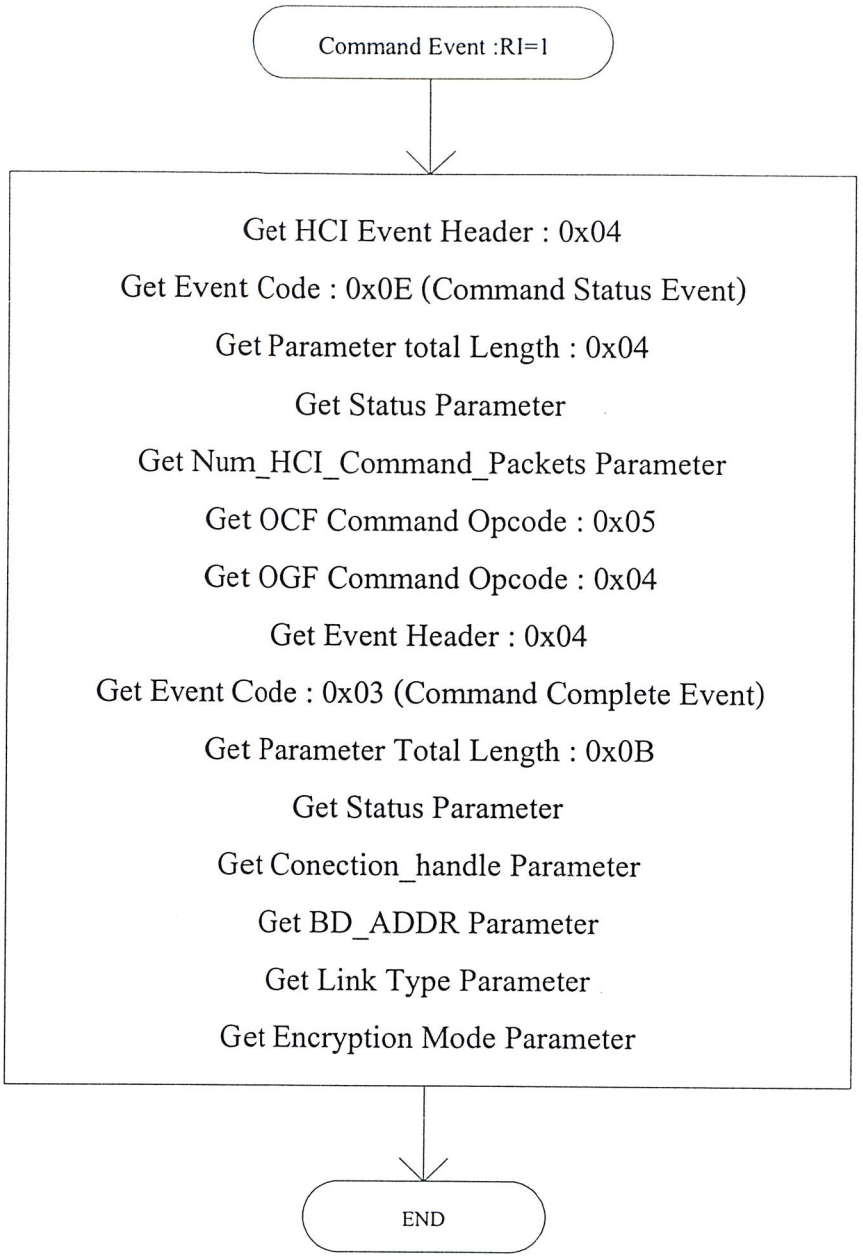



Command Event :RI=1
Get HCI Event Header : 0x04
Get Event Code : 0x0F (Command Status Event)
Get Parameter total Length : 0x04
Get Status Parameter
Get Num_HCI_Command_Packets Parameter
Get OCF Command Opcode : 0x01
Get OGF Command Opcode : 0x04
Get HCI Event Header : 0x04
Get Event Code : 0x02 (Inquiry Result Event)
Get Parameter total Length : 0x0F
Get Num_Response Parameter
Get BD_ADDR parameter
Get Page_Scan_Repetition_Mode[i] Parameter
Get page_Sgan_Period_Mode[i] Parameter
Get Page_Scan_mode[i] Parameter
Get Class_of_Device[i] Parameter
Get Clock_Offset[i] Parameter
Get HCI Event Header : 0x04
Get Event Code : 0x01 (Inquiry Complete Event)
Get Parameter Total Length : 0x01
Get Status Parameter
END

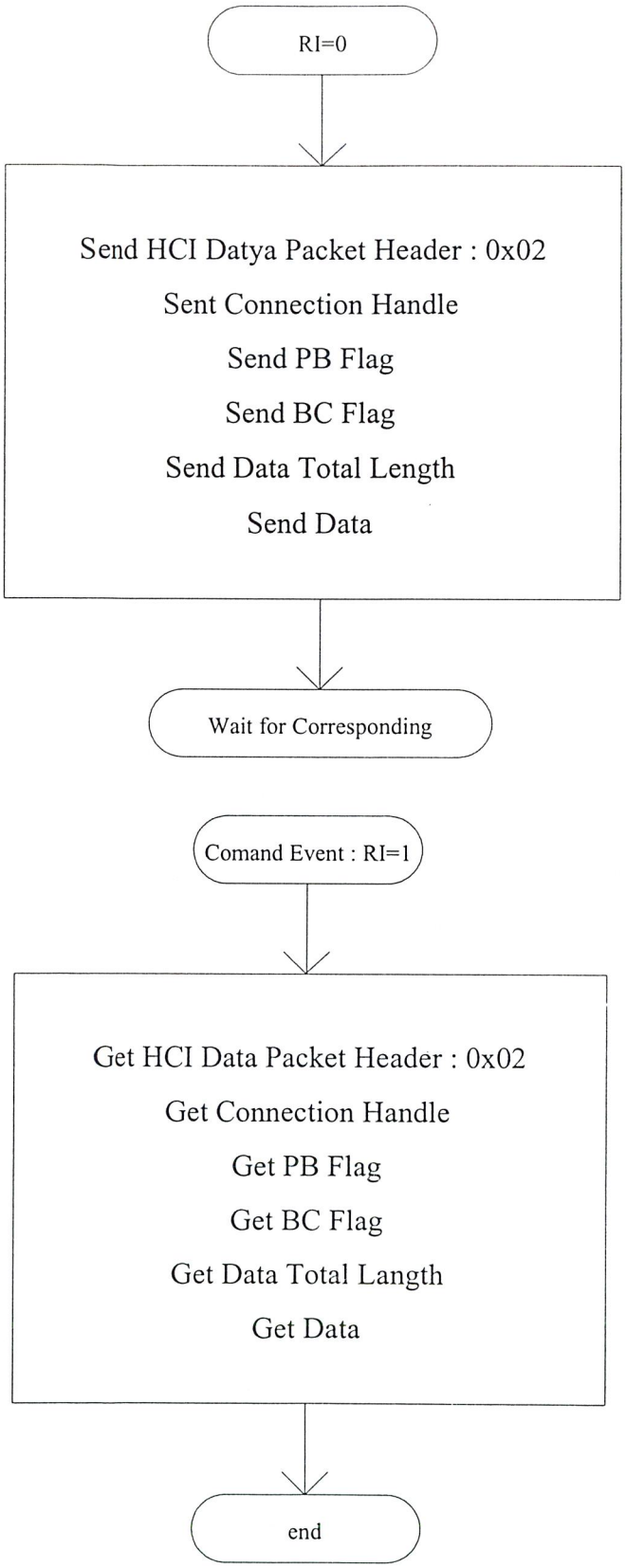
16. คำสั่งที่ใช้ในการขอการติดต่อ (Create_Connection) : OpCode = 0x0405

เป็นคำสั่งที่ทำให้ LM ติดต่อแบบ ACL ของอุปกรณ์บลูทูธ ถึง BD_ADDR



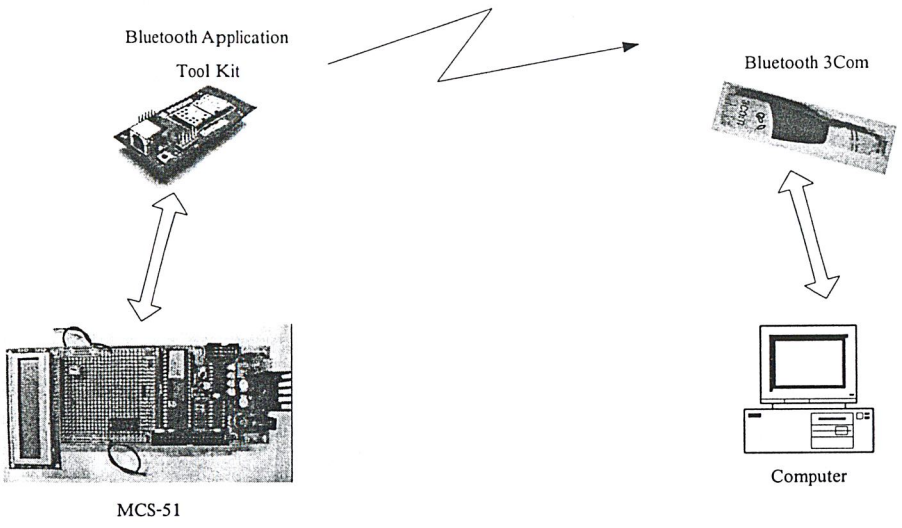

Command Event :RI=1
Get HCI Event Header : 0x04
Get Event Code : 0x0E (Command Status Event)
Get Parameter total Length : 0x04
Get Status Parameter
Get Num_HCI_Command_Packets Parameter
Get OCF Command Opcode : 0x05
Get OGF Command Opcode : 0x04
Get Event Header : 0x04
Get Event Code : 0x03 (Command Complete Event)
Get Parameter Total Length : 0x0B
Get Status Parameter
Get Conection_handle Parameter
Get BD_ADDR Parameter
Get Link Type Parameter
Get Encryption Mode Parameter
END

17. คำสั่งที่ใช้ในการส่งข้อมูล (ACL Data Packet)



3.3.4 ทำการเขียนโปรแกรมบนไมโครคอนโทรลเลอร์ด้วยภาษาซี (สามารถดูโค้ดได้ในภาคผนวก)

3.3.5 ต่ออุปกรณ์ดังรูปที่ 3.7 แล้วทำการค้นหาอุปกรณ์บลูทูธที่อยู่ข้างเคียงและทำการส่งข้อมูลจากไมโครคอนโทรลเลอร์ โดยส่งคำว่า "AAA" ไปยังคอมพิวเตอร์



รูปที่ 3.7 การต่ออุปกรณ์รวมทั้งหมด

บทที่ 4

ผลการทดลอง

ทำการทดลองให้ไมโครคอนโทรลเลอร์ส่งคำสั่งต่างๆตามโฟลชาร์ตแล้วรอรับแพ็กเกจที่ตอบกลับมาให้ไปแสดงผลที่จอแอลซีดี นำมาพิจารณาค่าที่ตอบกลับมาได้ดังนี้

4.1 คำสั่งที่ใช้ในการีเซต (Reset)

ส่งค่า : 01 03 0C 00

รูปที่ 4.1 แพ็กเกจที่ตอบกลับมาของคำสั่งรีเซต

จากการทดลองเป็นการส่งค่า การรีเซตไปยังบอร์ดบลูทูธ เพื่อเป็นการเตรียมรับข้อมูลใหม่ที่จะถูกส่งมา ซึ่งค่าที่ส่งไปยังบอร์ดบลูทูธ บลูทูธสามารถรับค่าเหล่านี้ได้เนื่องจากค่าพารามิเตอร์ที่ตอบกลับมาในไบท์ที่ 7 มีค่า 00 และมีค่า OpCode ตรงกันกับที่ส่ง (คือ 03 0C) ซึ่งเป็นค่าที่ยอมรับการติดต่อ

4.2 คำสั่งที่ใช้กำหนดขนาดของ Host (Host_Buffer_Size)

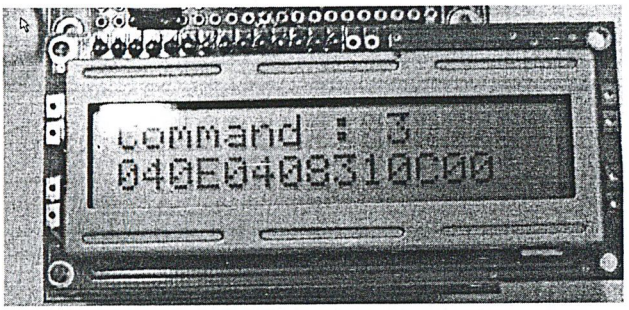
ส่งค่า : 01 33 0C 07 A0 02 00 0A 00 00 00

รูปที่ 4.2 แพ็กเกจที่ตอบกลับมาของคำสั่งกำหนดขนาดของ Host

จากการทดลองเป็นการส่งค่าที่กำหนดขนาดของบัพเฟอร์ของHost ไปยังบอร์ดบลูทูธโดยการส่งค่าไปยังบอร์ดบลูทูธ บลูทูธสามารถรับค่าเหล่านี้ได้เนื่องจากค่าพารามิเตอร์ที่ตอบกลับมาในไบท์ที่ 7 มีค่า 00 และมีค่า OpCode ตรงกันกับที่ส่ง (คือ 33 0C) ซึ่งเป็นค่าที่ยอมรับการติดต่อ

4.3 คำสั่งที่ใช้ในการควบคุมจาก Host ถึง Host Controller (Set_Host_Controller_to_Host_Flow_Control)

ส่งค่า : 01 31 0C 01 01

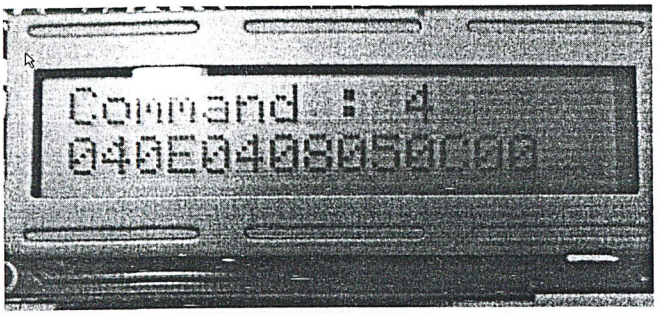


รูปที่ 4.3 แพ็กเกจที่ตอบกลับมาของคำสั่งที่ใช้ในการควบคุมจาก Host ถึง Host Controller

จากการทดลองเป็นการส่งค่าที่ใช้ในการควบคุมจากHostถึง Host Controller ไปยังบอร์ดบลูทูธ เพื่อเป็นการเชื่อมการติดต่อระหว่างบอร์ดบลูทูธกับไมโครคอนโทรลเลอร์ ซึ่งค่าที่ส่งไปยังบอร์ดบลูทูธ บลูทูธสามารถรับค่าเหล่านี้ได้เนื่องจากค่าพารามิเตอร์ที่ตอบกลับมาในไบท์ที่ 7 มีค่า 00 และและมีค่า OpCode ตรงกันกับที่ส่ง (คือ 31 0C) ซึ่งเป็นค่าที่ยอมรับการติดต่อ

4.4 คำสั่งที่ใช้ในการเซตค่า Fitler (Set_Event_Filter)

ส่งค่า : 01 05 0C 03 02 00 01

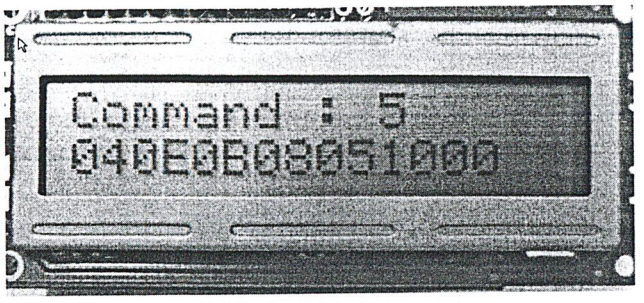


รูปที่ 4.4 แพ็กเกจที่ตอบกลับมาของคำสั่งที่ใช้ในการเซตค่า

จากการทดลองเป็นการส่งค่าที่ใช้ในการเซตค่า Filter ไปยังบอร์ดบลูทูธ ซึ่งค่าที่ส่งไปยังบอร์ดบลูทูธ บลูทูธสามารถรับค่าเหล่านี้ได้เนื่องจากค่าพารามิเตอร์ที่ตอบกลับมาในไบท์ที่ 7 มีค่า 00 และมีค่า OpCode ตรงกันกับที่ส่ง (คือ 05 0C) ซึ่งเป็นค่าที่ยอมรับการติดต่อ

4.5 คำสั่งที่ใช้ในการอ่านค่า Buffer (Read_Buffer_Size)

ส่งค่า : 01 05 10 00



รูปที่ 4.5 แพ็กเกจที่ตอบกลับมาของคำสั่งที่ใช้ในการอ่านค่า Buffer

จากการทดลองเป็นการส่งค่าที่ใช้ในการอ่านค่าบัฟเฟอร์ไปยังบอร์ดบลูทูธ เพื่อเป็นการอ่านขนาดบัฟเฟอร์ของบลูทูธ ซึ่งค่าที่ส่งไปยังบอร์ดบลูทูธ บลูทูธสามารถรับค่าเหล่านี้ได้เนื่องจากค่าพารามิเตอร์ที่ตอบกลับมาในไบท์ที่ 7 มีค่า 00 และมีค่า OpCode ตรงกันกับที่ส่ง (คือ 05 10) ซึ่งเป็นค่าที่ยอมรับการติดต่อ ส่วนค่าที่ตามมา A0 02 00 08 00 00 00 เป็นขนาดของบัฟเฟอร์ที่อ่านได้

4.6 คำสั่งที่ใช้ในการอ่านค่าแอดเดรส (Read_BD_ADDR)

ส่งค่า : 01 09 10 00

รูปที่ 4.6 แพ็กเกจที่ตอบกลับมาของคำสั่งที่ใช้ในการอ่านค่าอ่านค่าแอดเดรส

จากการทดลองเป็นการส่งค่าที่ใช้ในการอ่านค่าแอดเดรส ไปยังบอร์ดบลูทูธ เพื่อเป็นการอ่านค่าแอดเดรสของบลูทูธ ซึ่งค่าที่ส่งไปยังบอร์ดบลูทูธ บลูทูธสามารถรับค่าเหล่านี้ได้เนื่องจากค่าพารามิเตอร์ที่ตอบกลับมาในไบท์ที่ 7 มีค่า 00 ซึ่งเป็นค่าที่ยอมรับการติดต่อ ส่วนค่าที่ตามมา 8D 65 31 37 80 00 เป็นขนาดของแอดเดรสของบลูทูธที่อ่านได้

4.7 คำสั่งที่ใช้ในการอ่านค่าแอดเดรส (Read_BD_ADDR)

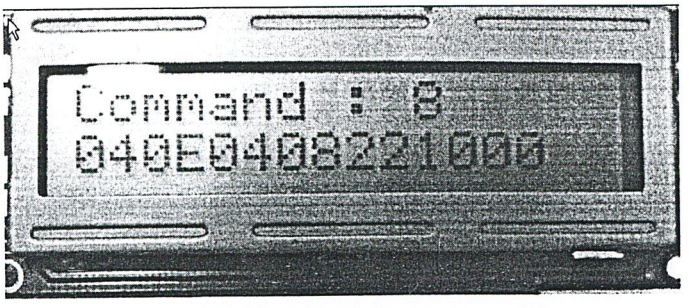
ส่งค่า : 01 09 10 00

รูปที่ 4.7 แพ็กเกจที่ตอบกลับมาของคำสั่งที่ใช้ในการอ่านค่า Buffer

จากการทดลองเป็นอ่านค่าแอแดรสไปยังบอร์ดบลูทูธอีกครั้ง เนื่องจากถ้าทำการอ่านครั้งเดียวค่าแอดแดสดที่อ่านได้จะเกิด error บลูทูธสามารถรับค่าเหล่านี้ได้เนื่องจากค่าพารามิเตอร์ที่ตอบกลับมาในไบท์ที่ 7 มีค่า 00 ซึ่งเป็นค่าที่ยอมรับการติดต่อ (ใช้โฟลชาร์ตเดียวกันกับคำสั่งที่ 6)

4.8 คำสั่งที่ใช้ในการยืนยันการทำงาน (Write_Authentication_Eanble)

ส่งค่า : 01 09 10 00

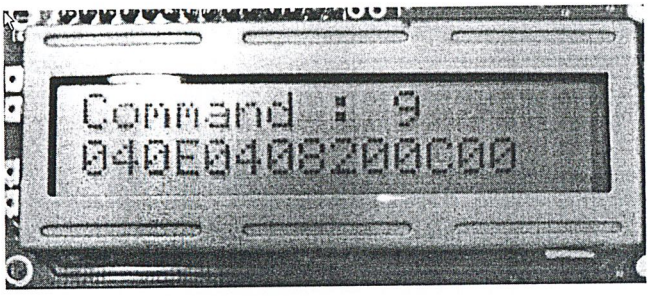


รูปที่ 4.8 แพ็กเกจที่ตอบกลับมาของคำสั่งที่ใช้ในการยืนยันการทำงาน

จากการทดลองเป็นการส่งค่าที่ใช้ในการยืนยันการทำงานไปยังบอร์ดบลูทูธ ซึ่งค่าที่ส่งไปยังบอร์ดบลูทูธ บลูทูธสามารถรับค่าเหล่านี้ได้เนื่องจากค่าพารามิเตอร์ที่ตอบกลับมาในไบท์ที่ 7 มีค่า 00 ซึ่งเป็นค่าที่ยอมรับการติดต่อ

4.9 คำสั่งที่ใช้ในการเขียนค่าของเวลาในการตอบรับก่อนหมดเวลา (Write_Connection_Accept_Timeout)

ส่งค่า : 01 20 0C 01 00



รูปที่ 4.9 แพ็กเกจที่ตอบกลับมาของคำสั่งที่ใช้ในการเขียนค่าของเวลาในการตอบรับก่อนหมดเวลา

จากการทดลองเป็นการส่งค่าที่ใช้ในการเขียนค่าของเวลาในการตอบรับก่อนหมดเวลา ซึ่งค่าที่ส่งไปยังบอร์ดบลูทูธ บลูทูธสามารถรับค่าเหล่านี้ได้เนื่องจากค่าพารามิเตอร์ที่ตอบกลับมาในไบท์ที่ 7 มีค่า 00 ซึ่งเป็นค่าที่ยอมรับการติดต่อ

4.10 คำสั่งที่ใช้ในการเขียนค่าของการเชื่อมต่อระหว่าง Layer Local Link Manager กับ Layer Baseband (Write_Page_Timeout)

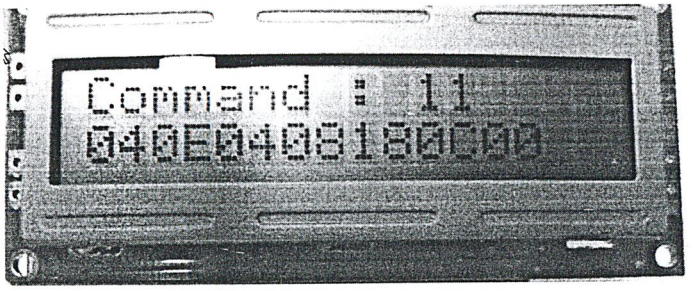
ส่งค่า : 01 10 0C 02 A0 1F

รูปที่ 4.10 แพ็กเกจที่ตอบกลับมาของ คำสั่งที่ใช้ในการเขียนค่าของการเชื่อมต่อระหว่าง Layer Local Link Manager กับ Layer Baseband

จากผลการทดลองเป็นการส่งค่าที่ใช้ในการเขียนค่าของการเชื่อมต่อระหว่าง Layer Local Link Manager กับ Layer Baseband ไปยังบอร์ดบลูทูธ เพื่อเป็นการเชื่อมต่อในการส่งข้อมูลของบลูทูธ ซึ่งค่าที่ส่งไปยังบอร์ดบลูทูธ บลูทูธสามารถรับค่าเหล่านี้ได้เนื่องจากค่าพารามิเตอร์ที่ตอบกลับมาในไบท์ที่ 7 มีค่า 00 ซึ่งเป็นค่าที่ยอมรับการติดต่อ

4.11 คำสั่งที่ใช้ในการเซตค่าของเสียง (Write_Voice_Setting)

ส่งค่า : 01 18 0C 02 40 1F

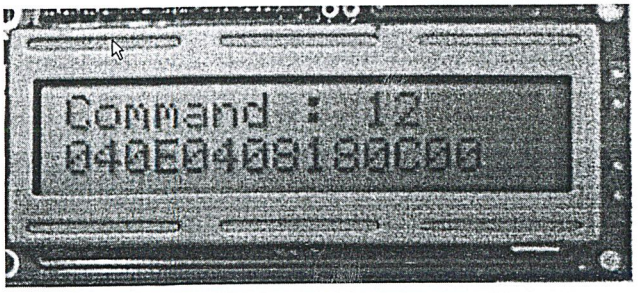


รูปที่ 4.11 แพ็กเกจที่ตอบกลับมาของคำสั่งที่ใช้ในการเซตค่าของเสียง

จากผลการทดลองเป็นการส่งค่าที่ใช้ในการเซตคาของเสียงไปยังบอร์ดบลูทูธ ซึ่งค่าที่ส่งไปยังบอร์ดบลูทูธ บลูทูธสามารถรับค่าเหล่านี้ได้เนื่องจากค่าพารามิเตอร์ที่ตอบกลับมาในไบท์ที่ 7 มีค่า 00 ซึ่งเป็นค่าที่ยอมรับการติดต่อ

4.12 คำสั่งที่ใช้ Layer ต่าง ๆ ของอุปกรณ์ (Write_Class_Of_Device)

ส่งค่า : 01 26 0C 02 62 00

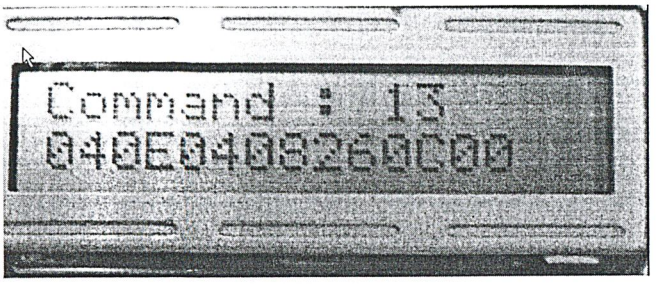


รูปที่ 4.12 แพ็กเกจที่ตอบกลับมาของ คำสั่งที่ใช้ Layer ต่าง ๆ ของอุปกรณ์

จากการทดลองเป็นการส่งค่าที่จะใช้เลเยอร์ตาง ๆ ของอุปกรณ์ ซึ่งค่าที่ส่งไปยังบอร์ดบลูทูธ บลูทูธสามารถรับค่าเหล่านี้ได้เนื่องจากค่าพารามิเตอร์ที่ตอบกลับมาในไบท์ที่ 7 มีค่า 00 ซึ่งเป็นค่าที่ยอมรับการติดต่อ

4.13 คำสั่งที่ใช้ในการเปลี่ยนชื่ออุปกรณ์เมื่อทำการติดต่อกันระหว่างอุปกรณ์ (Change_Local_Name)

ส่งค่า : 01 24 0C 03 04 04 04 20

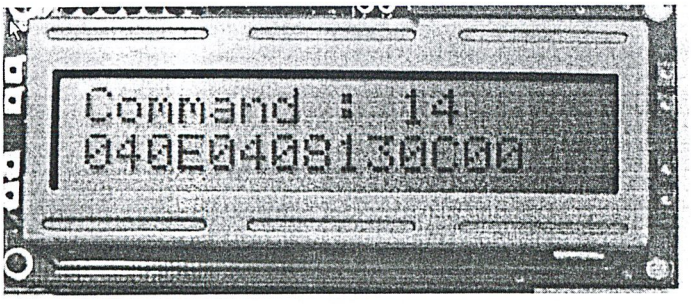


รูปที่ 4.13 แพ็กเกจที่ตอบกลับมาของคำสั่งที่ใช้ในการเปลี่ยนชื่ออุปกรณ์เมื่อทำการติดต่อกันระหว่างอุปกรณ์

จากการทดลองเป็นการส่งค่าที่ใช้ในการเปลี่ยนชื่อของอุปกรณ์เมื่อทำการติดต่อกันระหว่างอุปกรณ์ไปยังบอร์ดบลูทูธ ซึ่งค่าที่ส่งไปยังบอร์ดบลูทูธ บลูทูธสามารถรับค่าเหล่านี้ได้เนื่องจากค่าพารามิเตอร์ที่ตอบกลับมาในไบท์ที่ 7 มีค่า 00 ซึ่งเป็นค่าที่ยอมรับการติดต่อ

4.14 คำสั่งที่ใช้ในการส่งคลื่นสัญญาณของตัวอุปกรณ์ (Write_Scan_Enable)

ส่งค่า : 01 13 0C F8 42 54 20 43 68 61 74 00 00 00 00……..



รูปที่ 4.14 แพ็กเกจที่ตอบกลับมาของคำสั่งที่ใช้ในการส่งคลื่นสัญญาณของตัวอุปกรณ์

จากการทดลองเป็นการส่งค่าที่ใช้ในการส่งคลื่นสัญญาณของบอร์ดบลูทูธ ซึ่งค่าที่ส่งไปยังบอร์ดบลูทูธ บลูทูธสามารถรับค่าเหล่านี้ได้เนื่องจากค่าพารามิเตอร์ที่ตอบกลับมาในไบท์ที่ 7 มีค่า 00 ซึ่งเป็นค่าที่ยอมรับการติดต่อ ส่วนค่าที่ตามมา 04 04 0A 91 65 31 37 80 00 04 04 20 01 เป็นค่าแอดเดรสที่เกิดจากการส่งคลื่นสัญญาณออกไปแล้วพบตัวอุปกรณ์ที่เป็นบลูทูธด้วยกัน

4.15 คำสั่งที่ใช้ในการยอมรับการติดต่อ (Accept_Connection_Request)

ส่งค่า : 01 1A 0C 01 03

รูปที่ 4.15 แพ็กเกจที่ตอบกลับมาของคำสั่งที่ใช้ในการยอมรับการติดต่อ

จากการทดลองเป็นการส่งค่าที่ในการยอมรับการติดต่อซึ่งค่าที่ส่งไปยังบอร์ดบลูทูธ บลูทูธสามารถรับค่าเหล่านี้ได้เนื่องจากค่าพารามิเตอร์ที่ตอบกลับมาในไบท์ที่ 11 มีค่า 00 ซึ่งเป็นคาที่ยอมรับการติดต่อ ส่วนค่าที่ตามมา 01 00 91 65 31 37 80 00 01 00 เป็นค่าแอดเดรสของอุปกรณ์ที่เกิดจากการยอมรับการติดต่อ

4.16 คำสั่งที่ใช้ในการค้นหาอุปกรณ์รอบข้าง (Inquiry)

ส่งค่า : 01 01 05 33 8B 9E 02 00

รูปที่ 4. 16 แสดงแอดแดรสของอุปกรณ์บลูทูธรอบข้างที่หามาได้

เมื่อโปรแกรมรันมาถึงคำสั่งที่ 16 อุปกรณ์บลูทูธจะทำการค้นหาอุปกรณ์ใกล้เคียงและร้องขอ BD_ADDR จากผลการทดลองพบว่าอุปกรณ์บลูทูธสามารถค้นหาอุปกรณ์รอบข้างพบโดยพิจารณาจาก BD_ADDR ที่ได้ในแพ็กเกจที่ตอบกลับมาของอุปกรณ์บลูทูธว่ามีตรงกับอุปกรณ์บลูทูธที่ต่อกับคอมพิวเตอร์ (นำส่วนของค่า BD_ADDR ที่อยู่ในแพ็กเกจมาแสดงที่จอแอลซีดีเท่านั้น)

4.17 คำสั่งที่ใช้ในการขอการติดต่อ (Create_Connection)

ส่งค่า : 01 05 04 0D 69 B6 E1 76 04 00

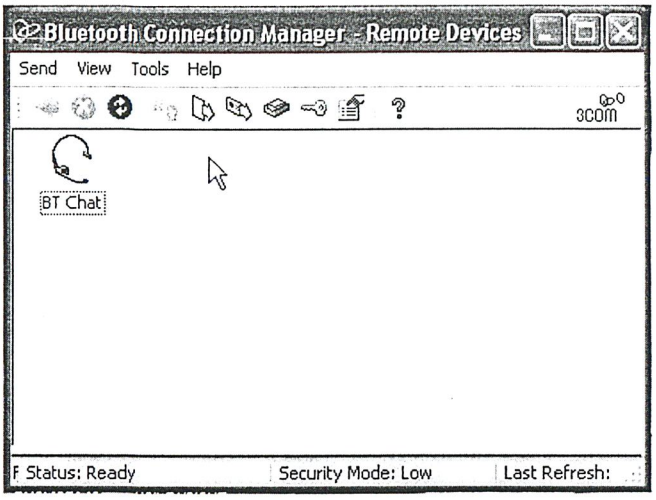
รูปที่ 4. 17 แสดงแอดแดรสของอุปกรณ์บลูทูธที่ติดต่อสำเร็จ

จากรูป 4.17 ผลการทดลองทำให้ทราบว่าอุปกรณ์บลูทูธที่ต่อกับคอมพิวเตอร์ยอมรับการติดต่อแล้ว เนื่องจากส่งค่า BD_ADDR มาให้อีกครั้ง

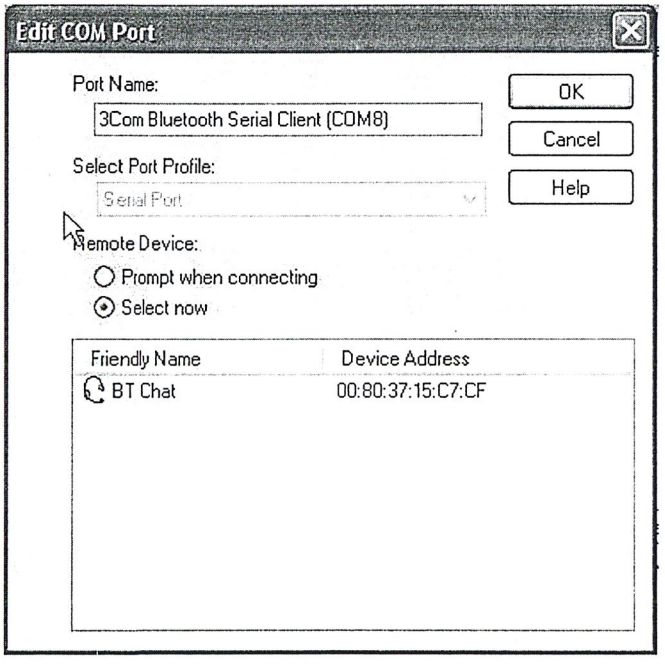
จากการทดลองสรุปได้ว่าโปรแกรมไมโครคอนโทรเลอร์ที่เขียนมาสามารถร้องขอการติดต่อได้ ต่อเป็นผลการทดลองในส่วนของให้บลูทูธที่ต่อกับไมโครคอนโทรเลอร์รอการร้องขอการติดต่อ

4.18 ผลการทดลองขณะที่บลูทูธที่ต่อกับไมโครคอนโทรลเลอร์รอการร้องขอการติดต่อ

ทำการรันโปรแกรมรีเซตที่ไมโครคอนโทรลเลอร์ แล้วทำการรันโปรแกรมที่เป็นแอบพลิเคชั่นของกรณ์บลูทูธ (3Com) อีกตัวที่ต่อกับคอมพิวเตอร์ อุปกรณ์บลูทูธจะทำการหาตัวข้างเคียงแล้วทำการเชื่อมต่อดังผลการทดลองดังรูป 4.18 และรูปที่ 4.19



รูปที่ 4.18 แสดงการเชื่อมต่อและหาอุปกรณ์ใกล้เคียงพบ



รูปที่ 4.19 แสดงค่า BD_ADDR ขออุปกรณ์บลูทูธที่ทำการเชื่อมต่อ

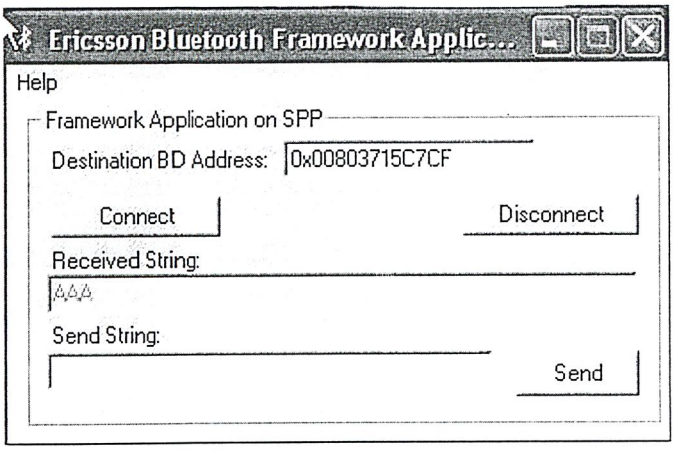
จากรูป 4.19 จะเห็นได้ว่าทำการทำการเชื่อมต่อกับอุปกรณ์บลูทูธที่ต่อกับไมโครคอนโทรเลอร์ได้สำเร็จแสดงได้จากค่า BD_ADDR ที่อ่านได้

4.19 ผลการทดลองในส่วนของการส่งข้อมูล

ทำการทดลองส่งข้อมูลจากไมโครคอนโทรเลอร์ไปคอมพิวเตอร์ โดยส่งคำว่า "AAA" ดังรูปที่ 4.20

รูปที่ 4.20 แสดงข้อมูลที่ใช้ส่งขณะทำการส่ง

จากการทดลองสามารถส่งข้อมูลเข้าคอมพิวเตอร์ได้ดังรูปที่ 4.21



รูปที่ 4.21 แสดงข้อมูลที่เครื่องคอมพิวเตอร์สามารถรับได้

บทที่ 5
สรุปและวิเคราะห์

5.1 สรุปผลการทดลอง

5.1.1 การทำงานของบลูทูธ แบ่งออกปีนเลเยอร์โดยถ้าจะทำการติดต่อหรือควบคุมอุปกรณ์บลูทูธสามารถทำได้ที่ชั้น HCI เลเยอร์

5.1.2 โครงงานนี้จึงได้ทำการส่งคำสั่งไปที่ HCI เลเยอร์ เพื่อทำการควบคุมให้อุปกรณ์บลูทูธทำงานตามต้องการได้

5.1.3 อุปกรณ์บลูทูธที่ใช้ในการทดลองสามารถติดต่อกันได้โดยไม่ต้องใช้คอมพิวเตอร์ควบคุม

5.1.4 สามารถใช้ไมโครคอนโทรลเลอร์ส่งข้อมูลไปยังคอมพิวเตอร์ โดยผ่านอุปกรณ์บลูทูธได้

5.1.5 อุปกรณ์บลูทูธส่งออกอากาศด้วยคลื่นความถี่ประมาณ 2.4 GHz

5.1.6 สามารถทำการค้นหาอุปกรณ์บลูทูธใกล้เคียงได้ถึง 8 เมตร

5.2 ปัญหาที่เกิดขึ้นจากการทดลอง

5.2.1 การศึกษาในด้านทฤษฎีของบลูทูธเป็นไปได้ยากเพราะส่วนใหญ่พบในลักษณะที่เป็นภาษาอังกฤษ

5.2.2 จากการทดลองพบว่าคำสั่งที่ใช้บางคำสั่งเหมือนกันแต่ต้องทำการส่งถึงสองครั้ง เช่น คำสั่งอ่านค่า BD_ADDR จึงจะสามารถเชื่อมต่อกันได้

5.3 แนวทางนำไปประยุกต์ใช้อนาคต

5.3.1 สามารถนำส่วนของการเขียนโปแกรมบนไมโครคอนโทรลเลอร์ไปประยุกต์ใช้ในการส่ง-รับข้อมูลการควบคุมอปกรณ์ไปยังอุปกรณ์ที่ไม่มีคอมพิวเตอร์แต่มีไมโครคอนโทรเลอร์ควบคุมแทนได้ เช่น หุ่นยนต์

5.3.2 สามารถนำความรู้ที่ได้เกี่ยวกับอุปกรณ์บลูทูธไปประยุกต์ใช้ในการส่งข้อมูลผ่านคอมพิวเตอร์ การทำระบบ LAN ไร้สาย อินเตอร์เนตไร้สายได้ในอนาคต เป็นต้น

บรรณานุกรม

[1] Jennifer Bray, Charles F. Sturmam, 2001, "Bluetooth Connect Without Cables", 1st edition, Prentice Hall PTR, United States of America, page 1 - 191

[2] ลภน สุภาพ, ปีที่พิมพ์ 2545, "Bluetooth เทคโนโลยีการเชื่อมต่ออุปกรณ์แบบไร้สาย ", เซมิคอนดักเตอร์อิเล็กทรอนิกส์, ปีที่ 2545, ฉบับที่ 234, หน้า 184 – 192

[3] ลภน สุภาพ, ปีที่พิมพ์ 2545, "ผ่าเทคโนโลยีอุปกรณ์ Bluetooth ", เซมิคอนดักเตอร์อิเล็กทรอนิกส์ , ปีที่ 2545, ฉบับที่ 234, หน้า 213 – 221

[4] พงษ์ศักดิ์ สุสัมพันธ์ไพบูลย์, ปีที่พิมพ์ 2543, "Bluetooth เทคโนโลยีสื่อสารไร้สาย", สาร NECTEC 7, ปีที่ 2543, ฉบับที่ 36, หน้า 73 – 82

[5] รองศาสตราจารย์ มัณฑนา ปราการสมุทร , การเขียนชุดคำสั่งภาษาซี , ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย , กรุงเทพ , 2545

[6] พัฒนพล สายกลิ่น และ อาจารย์มารุต ตามศรี , การพัฒนาไมโครคอนโทรลเลอร์ด้วยภาษาซี , บริษัท ศิลารีเซิร์ท , กรุงเทพ , 2540

[7] วรพจน์ กรแก้ววัฒนกุล , เรียนรู้และปฏิบัติการไมโครคอนโทรลเลอร์ , INEX , กรุงเทพ

ภาคผนวก

ส่วนของโปรแกรมที่ใช้ในการส่งข้อมูล

```
#include <89c51rd2.h>
#include <printf.h>

unsigned char SendToBuf[45];
unsigned char ReceivePkt[275];
int t,r,n=0;

void Send_ACL_Packet(unsigned char C_PB,unsigned char C_BC,
unsigned char PTotalLength_1,unsigned char PTotalLength_2,int PacketNO);
void ACL_CommandParameter(int PacketNO);
void Check_inquiry(void);
void loop(void);

struct {
   unsigned char  PacketType;
   unsigned char  OpCodeH;
   unsigned char  OpCodeL;
   unsigned char ParameterTotalLength;
   unsigned char Parameter[10];
} CommandPacket;

struct {
   unsigned char  ACLPacketType;
   unsigned char  C_PB;
   unsigned char  C_BC;
   unsigned char PTotalLength_1;
   unsigned char PTotalLength_2;
} ACLpacket;
```

```
void serial_init(void)
{
        SCON = 0x50;
        TCLK = 0x01;
        RCLK = 0x01;
        TH2 = 0xFF;
        TL2 = 0xDD;
        RCAP2H = 0xFF;
        RCAP2L = 0xEC;
   TR2 = 1;
        EA = 1;
        ES = 1;
        EX1 = 1;
}

void serial(void) interrupt 4 using 2  {   // use registerbank 2 for interrupt
 if(RI){                    // if receiver interrupt
        RI = 0;             // clear interrupt request flag
        r  = 1;
        ReceivePkt[n] = SBUF;
        n++;
 }
 if(TI){                    // if transmitter interrupt
        TI = 0;             // clear interrupt request flag
        t = 1;
 }
}
```

```
        i = 0;
         while(i<1){
                SBUF = SendToBuf[i];
                i++;
    while(t!=1);
                 t=0;
                }break;

case 4:
        SendToBuf[0] = 0x02;
        SendToBuf[1] = 0x00;
        SendToBuf[2] = 0x01;
        i = 0;
         while(i<3){
                SBUF = SendToBuf[i];
                i++;
                 while(t!=1);
                  t=0;
                }break;

case 8:
        SendToBuf[0] = 0x00;
        i = 0;
         while(i<1){
                SBUF = SendToBuf[i];
                i++;
                while(t!=1);
                 t=0;
                }break;

case 9:
```

```
void delays(unsigned int count)
{
        unsigned int i,j;
                for (i=0;i<count;i++)
                for (j=0;j<1000;j++);
}
void CommandParameter(int PacketNO)
{
 int i = 0;
 switch(PacketNO)
 {
  case 2:
          SendToBuf[0] = 0xA0;
          SendToBuf[1] = 0x02;
          SendToBuf[2] = 0x00;
          SendToBuf[3] = 0x0A;
          SendToBuf[4] = 0x00;
          SendToBuf[5] = 0x00;
          SendToBuf[6] = 0x00;
          i = 0;
           while(i<7){
            SBUF = SendToBuf[i];
            i++;
             while(t!=1);
                    t=0;
           }break;

   case 3:
          SendToBuf[0] = 0x01;
```

```
        SendToBuf[0] = 0x00;
         while(i<1){
                SBUF = SendToBuf[i];
                i++;
                 while(t!=1);
                  t=0;
                }break;

case 10:
        SendToBuf[0] = 0xA0;
        SendToBuf[1] = 0x1F;
        i = 0;
          while(i<2){
                SBUF = SendToBuf[i];
                i++;
                 while(t!=1);
                   t=0;
                }break;

case 11:
        SendToBuf[0] = 0x40;
        SendToBuf[1] = 0x1F;
        i = 0;
          while(i<2){
                SBUF = SendToBuf[i];
                i++;
                 while(t!=1);
                   t=0;
                }break;

case 12:
```

```
        SendToBuf[0] = 0x62;
        SendToBuf[1] = 0x00;
        i = 0;
         while(i<2){
                SBUF = SendToBuf[i];
                i++;
                 while(t!=1);
                   t=0;
                }break;

case 13:
        SendToBuf[0] = 0x04;
        SendToBuf[1] = 0x04;
        SendToBuf[2] = 0x20;
        i = 0;
          while(i<3){
                 SBUF = SendToBuf[i];
                 i++;
                  while(t!=1);
                    t=0;
                 }break;


 case 14:
          SendToBuf[0] = 'B';
          SendToBuf[1] = 'T';
          SendToBuf[2] = ' ';
          SendToBuf[3] = 'C';
          SendToBuf[4] = 'h';
          SendToBuf[5] = 'a';
          SendToBuf[6] = 't';
```

```
        while(i<7){
                SBUF = SendToBuf[i];
                i++;
                 while(t!=1);
                  t=0;
                 }
                 i = 0;
                 while(i<241){
                  SBUF = 0x00;
                  i++;
                 while(t!=1);
                  t=0;
                 }break;

case 15:
        SendToBuf[0] = 0x03;
        i = 0;
         while(i<1){
                 SBUF = SendToBuf[i];
                 i++;
                 while(t!=1);
                 t=0;
                 }break;

 case 16:
         SendToBuf[0] = 0x33;
         SendToBuf[1] = 0x8B;
         SendToBuf[2] = 0x9E;
         SendToBuf[3] = 0x02;
         SendToBuf[4] = 0x00;
         i = 0;
```

```
        while(i<5){
                SBUF = SendToBuf[i];
                i++;
                while(t!=1);
                t=0;
                }break;

case 17:
        SendToBuf[0] = 0x69;
        SendToBuf[1] = 0xB6;
        SendToBuf[2] = 0xE1;
        SendToBuf[3] = 0x76;
        SendToBuf[4] = 0x04;
        SendToBuf[5] = 0x00;
        SendToBuf[6] = 0x00;
        SendToBuf[7] = 0x00;
        SendToBuf[8] = 0x1F;
        SendToBuf[9] = 0x67;
 i = 0;
         while(i<10){
                SBUF = SendToBuf[i];
                i++;
                while(t!=1);
                t=0;
                }break;

case 18:
        SendToBuf[0] = 0x69;
        SendToBuf[1] = 0xB6;
        SendToBuf[2] = 0xE1;
        SendToBuf[3] = 0x76;
```

```
        SendToBuf[4] = 0x04;
        SendToBuf[5] = 0x00;
        SendToBuf[6] = 0x08;
        SendToBuf[7] = 0x00;
        SendToBuf[8] = 0x01;
        SendToBuf[9] = 0x00;
        SendToBuf[10] = 0x00;
        SendToBuf[11] = 0x00;
        SendToBuf[12] = 0x00;
i = 0;
         while(i<13){
                SBUF = SendToBuf[i];
                i++;
                while(t!=1);
                t=0;
                }break;

case 23:
        SendToBuf[0] = 0x01;
        SendToBuf[1] = 0x01;
        SendToBuf[2] = 0x00;
        SendToBuf[3] = 0x05;
        SendToBuf[4] = 0x00;
        SendToBuf[5] = 0x02;
        SendToBuf[6] = 0x01;
        SendToBuf[7] = 0x20;
        SendToBuf[8] = 0x18;
        SendToBuf[9] = 0x00;
        SendToBuf[10] = 0x14;
        SendToBuf[11] = 0x00;
        SendToBuf[12] = 0x40;
```

```
SendToBuf[13] = 0x00;
SendToBuf[14] = 0x04;
SendToBuf[15] = 0x00;
SendToBuf[16] = 0x01;
SendToBuf[17] = 0x00;
SendToBuf[18] = 0x0F;
SendToBuf[19] = 0x00;
SendToBuf[20] = 0x01;
SendToBuf[21] = 0x00;
SendToBuf[22] = 0x00;
SendToBuf[23] = 0x02;
SendToBuf[24] = 0xA0;
SendToBuf[25] = 0x35;
SendToBuf[26] = 0x06;
SendToBuf[27] = 0x09;
SendToBuf[28] = 0x00;
SendToBuf[29] = 0x00;
SendToBuf[30] = 0x09;
SendToBuf[31] = 0x00;
SendToBuf[32] = 0x04;
SendToBuf[33] = 0x00;
i = 0;
 while(i<34){
        SBUF = SendToBuf[i];
        i++;
        while(t!=1);
        t=0;
        }break;

case 27:
    SendToBuf[0] = 0x01;
```

```
SendToBuf[1] = 0x01;
SendToBuf[2] = 0x00;
SendToBuf[3] = 0x05;
SendToBuf[4] = 0x00;
SendToBuf[5] = 0x02;
SendToBuf[6] = 0x01;
SendToBuf[7] = 0x20;
SendToBuf[8] = 0x0E;
SendToBuf[9] = 0x00;
SendToBuf[10] = 0x0A;
SendToBuf[11] = 0x00;
SendToBuf[12] = 0x01;
SendToBuf[13] = 0x00;
SendToBuf[14] = 0x05;
SendToBuf[15] = 0x02;
SendToBuf[16] = 0x06;
SendToBuf[17] = 0x00;
SendToBuf[18] = 0x40;
SendToBuf[19] = 0x00;
SendToBuf[20] = 0x00;
SendToBuf[21] = 0x00;
SendToBuf[22] = 0x00;
SendToBuf[23] = 0x00;
SendToBuf[24] = 0x02;
SendToBuf[25] = 0x01;
SendToBuf[26] = 0x20;
SendToBuf[27] = 0x08;
SendToBuf[28] = 0x00;
SendToBuf[29] = 0x04;
SendToBuf[30] = 0x00;
SendToBuf[31] = 0x40;
```

```
        SendToBuf[32] = 0x00;
        SendToBuf[33] = 0x03;
        SendToBuf[34] = 0x3F;
        SendToBuf[35] = 0x01;
        SendToBuf[36] = 0x1C;
        i = 0;
         while(i<37){
                SBUF = SendToBuf[i];
                i++;
                while(t!=1);
                t=0;
                }break;
        }
}

void SendCommandPacket(unsigned char OpCodeH,unsigned char OpCodeL,unsigned char
PTotalLength,int PacketNO){
 int i = 0;
        CommandPacket.PacketType = 0x01;
        CommandPacket.OpCodeH = OpCodeH;
        CommandPacket.OpCodeL = OpCodeL;
        CommandPacket.ParameterTotalLength = PTotalLength;
        SendToBuf[0] = CommandPacket.PacketType;
        SendToBuf[1] = CommandPacket.OpCodeH;
        SendToBuf[2] = CommandPacket.OpCodeL;
        SendToBuf[3] = CommandPacket.ParameterTotalLength;
        while(i<4){
         SBUF = SendToBuf[i];
         i++;
         while(t!=1);
          t=0;
```

```
        }
        if(PTotalLength!=0x00)
{
        switch(PacketNO)
        {
                case 2:  CommandParameter(2);break;
                case 3:  CommandParameter(3);break;
                case 4:  CommandParameter(4);break;
                case 8:  CommandParameter(8);break;
                case 9:  CommandParameter(9);break;
                case 10: CommandParameter(10);break;
                case 11: CommandParameter(11);break;
                case 12: CommandParameter(12);break;
                case 13: CommandParameter(13);break;
                case 14: CommandParameter(14);break;
                case 15: CommandParameter(15);break;
                case 16: CommandParameter(16);break;
                case 17: CommandParameter(17);break;
                case 18: CommandParameter(18);break;
                case 23: CommandParameter(23);break;
                case 27: CommandParameter(27);break;
        }
        }
 n = 0;
}

void loop(void)
{
 int x=1;
  while(x)
  {
```

```
        SendCommandPacket(0x01,0x04,0x05,16);//Inquiry
        delays(3000);
         if(ReceivePkt[11]==0x69)
         {
                x=0;
  }
}
}

void main (void)
{
        P2=0x00;
        serial_init();                                  // Set Buad_Rate 57600 Bps
        lcd_init();
        cls();
        line1();printf("transmit");
        line2();printf("AAA");
        delays(500);
        cls();

line1();printf("Command : 1");
SendCommandPacket(0x03,0x0C,0x00,1);   //Reset_Command
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[0],ReceivePkt[1],
ReceivePkt[2],ReceivePkt[3],ReceivePkt[4],ReceivePkt[5],ReceivePkt[6]);
delays(1000);

line1();printf("Command : 2");
SendCommandPacket(0x33,0x0C,0x07,2);   //Host_Buffer_Size
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[0],ReceivePkt[1],
ReceivePkt[2],ReceivePkt[3],ReceivePkt[4],ReceivePkt[5],ReceivePkt[6]);
delays(1000);
```

```
line1();printf("Command : 3");
SendCommandPacket(0x31,0x0C,0x01,3);   //Set_Host_controller_To_Host_Flow_Control
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[0],ReceivePkt[1],
ReceivePkt[2],ReceivePkt[3],ReceivePkt[4],ReceivePkt[5],ReceivePkt[6]);
delays(1000);

 line1();printf("Command : 4");
 SendCommandPacket(0x05,0x0C,0x03,4);  //Set_Event_Filter
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[0],ReceivePkt[1],
ReceivePkt[2],ReceivePkt[3],ReceivePkt[4],ReceivePkt[5],ReceivePkt[6]);
delays(1000);

 line1();printf("Command : 5");
SendCommandPacket(0x05,0x10,0x00,5);   //Read_Buffer_Size
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[0],ReceivePkt[1],
ReceivePkt[2],ReceivePkt[3],ReceivePkt[4],ReceivePkt[5],ReceivePkt[6]);
delays(1000);
cls();

 line1();printf("Command : 6");
 SendCommandPacket(0x09,0x10,0x00,6);   //Read_BD_ADDR
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[0],ReceivePkt[1],
  ReceivePkt[2],ReceivePkt[3],ReceivePkt[4],ReceivePkt[5],ReceivePkt[6]);
  delays(1000);

  line1();printf("Command : 7");
 SendCommandPacket(0x09,0x10,0x00,7);   //Read_BD_ADDR
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[7],ReceivePkt[8],ReceivePkt[9],
ReceivePkt[10],ReceivePkt[11],ReceivePkt[12]);
```

```
delays(1000);

line1();printf("Command : 8");
SendCommandPacket(0x22,0x0C,0x01,8);   //Write_Encrytion_Mode
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[0],ReceivePkt[1],
ReceivePkt[2],ReceivePkt[3],ReceivePkt[4],ReceivePkt[5],ReceivePkt[6]);
delays(1000);

 line1();printf("Command : 9");
 SendCommandPacket(0x20,0x0C,0x01,9);  //Write_Authentication_Enable
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[0],ReceivePkt[1],
  ReceivePkt[2],ReceivePkt[3],ReceivePkt[4],ReceivePkt[5],ReceivePkt[6]);
  delays(1000);

  line1();printf("Command : 10");
  SendCommandPacket(0x16,0x0C,0x02,10);          //Write_Connection_Accept_Time_Out
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[0],ReceivePkt[1],
ReceivePkt[2],ReceivePkt[3],ReceivePkt[4],ReceivePkt[5],ReceivePkt[6]);
          delays(1000);

    line1();printf("Command : 11");
   SendCommandPacket(0x18,0x0C,0x02,11);          //Write_Page_Time_Out

line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[0],ReceivePkt[1],Receive
Pkt[2],ReceivePkt[3],ReceivePkt[4],ReceivePkt[5],ReceivePkt[6]);
          delays(1000);

 line1();printf("Command : 12");
SendCommandPacket(0x26,0x0C,0x02,12);  //Write_Voice_Setting
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[0],ReceivePkt[1],Receive
Pkt[2],ReceivePkt[3],ReceivePkt[4],ReceivePkt[5],ReceivePkt[6]);
```

```
delays(1000);

line1();printf("Command : 13");
SendCommandPacket(0x24,0x0C,0x03,13);  //Write_Class_OF_Device
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[0],ReceivePkt[1],
ReceivePkt[2],ReceivePkt[3],ReceivePkt[4],ReceivePkt[5],ReceivePkt[6]);
delays(1000);

 line1();printf("Command : 14");
 SendCommandPacket(0x13,0x0C,0xF8,14); //Change_Local_Name
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[0],ReceivePkt[1],
ReceivePkt[2],ReceivePkt[3],ReceivePkt[4],ReceivePkt[5],ReceivePkt[6]);
delays(1000);

 line1();printf("Command : 15");
 SendCommandPacket(0x1A,0x0C,0x01,15);//Write_Scan_Enable
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[0],ReceivePkt[1],
ReceivePkt[2],ReceivePkt[3],ReceivePkt[4],ReceivePkt[5],ReceivePkt[6]);
delays(2500);
cls();

 line1();printf("Command : 16");
 loop();                    // command 16 Inquiry Address Bluetooth
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[11],ReceivePkt[12],
ReceivePkt[13],ReceivePkt[14],ReceivePkt[15],ReceivePkt[16]);
 delays(2500);

  line1();printf("Command : 18");
  SendCommandPacket(0x05,0x04,0x0D,18);          // command 18  Create_connection
line2();printf("%02bX%02bX%02bX%02bX%02bX%02bX",ReceivePkt[11],ReceivePkt[12],
  ReceivePkt[13],ReceivePkt[14],ReceivePkt[15],ReceivePkt[16]);
```

```
 delays(2500);

        while(1);
        {
                Send_ACL_Packet(0x01,0x20,0x0D,0x00,29);        // Send DATA
                delays(1000);
                }
        }
//END PROGRAM

void Check_inquiry(void)
{
        SendCommandPacket(0x19,0x04,0x0A,17); // Remote_Name_Request
        }


void Send_ACL_Packet(unsigned char C_PB,unsigned char C_BC,unsigned char PTotalLength_1,unsigned
char PTotalLength_2,int PacketNO){
 int i = 0;
        ACLpacket.ACLPacketType = 0x02;
        ACLpacket.C_PB = C_PB;
        ACLpacket.C_BC = C_BC;
        ACLpacket.PTotalLength_1 = PTotalLength_1;
        ACLpacket.PTotalLength_2 = PTotalLength_2;
        SendToBuf[0] = ACLpacket.ACLPacketType;
        SendToBuf[1] = ACLpacket.C_PB;
        SendToBuf[2] = ACLpacket.C_BC;
        SendToBuf[3] = ACLpacket.PTotalLength_1;
        SendToBuf[4] = ACLpacket.PTotalLength_2;
while(i<5){
  SBUF = SendToBuf[i];
```

```
  i++;
   while(t!=1);
          t=0;
   }
  if(PTotalLength_1!=0x00){
          switch(PacketNO){
                  case 19: ACL_CommandParameter(19);break;
                  case 20: ACL_CommandParameter(20);break;
                  case 21: ACL_CommandParameter(21);break;
                  case 22: ACL_CommandParameter(22);break;
                  case 24: ACL_CommandParameter(24);break;
                  case 25: ACL_CommandParameter(25);break;
                  case 26: ACL_CommandParameter(26);break;
                  case 28: ACL_CommandParameter(28);break;
                  case 29: ACL_CommandParameter(29);break;
          }
   }
}

void ACL_CommandParameter(int PacketNO){
  int i = 0;
switch(PacketNO){
          case 19:
                  SendToBuf[0] = 0x08;
                  SendToBuf[1] = 0x00;
                  SendToBuf[2] = 0x01;
                  SendToBuf[3] = 0x00;
                  SendToBuf[4] = 0x02;
                  SendToBuf[5] = 0x01;
                  SendToBuf[6] = 0x04;
                  SendToBuf[7] = 0x00;
```

```
SendToBuf[8] = 0x01;
SendToBuf[9] = 0x00;
SendToBuf[10] = 0x40;
SendToBuf[11] = 0x00;
i = 0;
while(i<12){
        SBUF = SendToBuf[i];
        i++;
        while(t!=1);
        t=0;
}break;

case 20:
        SendToBuf[0] = 0x08;
        SendToBuf[1] = 0x00;
        SendToBuf[2] = 0x01;
        SendToBuf[3] = 0x00;
        SendToBuf[4] = 0x04;
        SendToBuf[5] = 0x02;
        SendToBuf[6] = 0x04;
        SendToBuf[7] = 0x00;
        SendToBuf[8] = 0x40;
        SendToBuf[9] = 0x00;
        SendToBuf[10] = 0x00;
        SendToBuf[11] = 0x00;
        i = 0;
        while(i<12){
                SBUF = SendToBuf[i];
                i++;
                while(t!=1);
                t=0;
```

```
        }break;

case 21:
        SendToBuf[0] = 0x0A;
        SendToBuf[1] = 0x00;
        SendToBuf[2] = 0x01;
        SendToBuf[3] = 0x00;
        SendToBuf[4] = 0x05;
        SendToBuf[5] = 0x01;
        SendToBuf[6] = 0x06;
        SendToBuf[7] = 0x00;
        SendToBuf[8] = 0x40;
        SendToBuf[9] = 0x00;
        SendToBuf[10] = 0x00;
        SendToBuf[11] = 0x00;
        SendToBuf[12] = 0x00;
        SendToBuf[13] = 0x00;
        SendToBuf[14] = 0x02;
        SendToBuf[15] = 0x01;
        SendToBuf[16] = 0x20;
        SendToBuf[17] = 0x11;
        SendToBuf[18] = 0x00;
        SendToBuf[19] = 0x0D;
        SendToBuf[20] = 0x00;
        SendToBuf[21] = 0x40;
        SendToBuf[22] = 0x00;
        SendToBuf[23] = 0x02;
        SendToBuf[24] = 0x00;
        SendToBuf[25] = 0x01;
        SendToBuf[26] = 0x00;
        SendToBuf[27] = 0x08;
```

```
        SendToBuf[28] = 0x35;
        SendToBuf[29] = 0x03;
        SendToBuf[30] = 0x19;
        SendToBuf[31] = 0x11;
        SendToBuf[32] = 0x01;
        SendToBuf[33] = 0x00;
        SendToBuf[34] = 0x06;
        SendToBuf[35] = 0x00;
        i = 0;
        while(i<36){
                SBUF = SendToBuf[i];
                i++;
                while(t!=1);
                t=0;
        }break;

case 22:
        SendToBuf[0] = 0x11;
        SendToBuf[1] = 0x00;
        SendToBuf[2] = 0x40;
        SendToBuf[3] = 0x00;
        SendToBuf[4] = 0x04;
        SendToBuf[5] = 0x00;
        SendToBuf[6] = 0x01;
        SendToBuf[7] = 0x00;
        SendToBuf[8] = 0x0C;
        SendToBuf[9] = 0x00;
        SendToBuf[10] = 0x01;
        SendToBuf[11] = 0x00;
        SendToBuf[12] = 0x00;
        SendToBuf[13] = 0x02;
```

```
                SendToBuf[14] = 0xA0;
                SendToBuf[15] = 0x35;
                SendToBuf[16] = 0x03;
                SendToBuf[17] = 0x09;
                SendToBuf[18] = 0x01;
                SendToBuf[19] = 0x00;
                SendToBuf[20] = 0x00;
i = 0;
                while(i<21){
                        SBUF = SendToBuf[i];
                        i++;
                        while(t!=1);
                        t=0;
                }break;

    case 24:
                SendToBuf[0] = 0x08;
                SendToBuf[1] = 0x00;
                SendToBuf[2] = 0x01;
                SendToBuf[3] = 0x00;
                SendToBuf[4] = 0x06;
                SendToBuf[5] = 0x03;
                SendToBuf[6] = 0x04;
                SendToBuf[7] = 0x00;
                SendToBuf[8] = 0x40;
                SendToBuf[9] = 0x00;
                SendToBuf[10] = 0x40;
                SendToBuf[11] = 0x00;
                i = 0;
                while(i<12){
                        SBUF = SendToBuf[i];
```

```
            i++;
            while(t!=1);
            t=0;
        }break;

case 25:
        SendToBuf[0] = 0x08;
        SendToBuf[1] = 0x00;
        SendToBuf[2] = 0x01;
        SendToBuf[3] = 0x00;
        SendToBuf[4] = 0x02;
        SendToBuf[5] = 0x04;
        SendToBuf[6] = 0x04;
        SendToBuf[7] = 0x00;
        SendToBuf[8] = 0x03;
        SendToBuf[9] = 0x00;
        SendToBuf[10] = 0x40;
        SendToBuf[11] = 0x00;
        i = 0;
        while(i<12){
            SBUF = SendToBuf[i];
            i++;
            while(t!=1);
            t=0;
        }break;

case 26:
        SendToBuf[0] = 0x0C;
        SendToBuf[1] = 0x00;
        SendToBuf[2] = 0x01;
```

```
SendToBuf[3] = 0x00;
SendToBuf[4] = 0x04;
SendToBuf[5] = 0x05;
SendToBuf[6] = 0x08;
SendToBuf[7] = 0x00;
SendToBuf[8] = 0x40;
SendToBuf[9] = 0x00;
SendToBuf[10] = 0x00;
SendToBuf[11] = 0x00;
SendToBuf[12] = 0x01;
SendToBuf[13] = 0x02;
SendToBuf[14] = 0x84;
SendToBuf[15] = 0x00;
i = 0;
while(i<16){
        SBUF = SendToBuf[i];
        i++;
        while(t!=1);
        t=0;
}break;

case 29:
    SendToBuf[0] = 0x09;
    SendToBuf[1] = 0x00;
    SendToBuf[2] = 0x40;
    SendToBuf[3] = 0x00;
    SendToBuf[4] = 0x0B;
    SendToBuf[5] = 0xEF;
    SendToBuf[6] = 0x08;
    SendToBuf[7] = 0x00;
    SendToBuf[8] = 0x41;          //A -- DATA
```

```
                SendToBuf[9] = 0x41;            //A -- DATA
                SendToBuf[10] =0x51;            //A -- DATA
                SendToBuf[11] = 0x00;
                SendToBuf[12] = 0x9A;
        i = 0;
                while(i<13){
                        SBUF = SendToBuf[i];
                        i++;
                        while(t!=1);
                        t=0;
                }break;
    }
}
```

ส่วนของโปรแกรมที่ใช้กับจอแอลซีดี

```
#include <stdio.h>
#define d_lcd P0

sbit RS = P0^2;
sbit EN = P0^3;
//sbit RW = P3^5;

//char code reserve [3] _at_ 0x23;

void delay(unsigned int count){
unsigned int loop;
for(loop=0;loop<count;loop++){}
}
void lcd_enable(void){
EN = 0;
delay(200);
EN = 1;
delay(200);
}
void write_instruct(unsigned char command){
RS = 0;
d_lcd = (d_lcd & 0x0F)|(command & 0xF0);
lcd_enable();
d_lcd = (d_lcd & 0x0F)|((command<<4) & 0xF0);
lcd_enable();
}
void lcd_putc(unsigned char ascii){
RS = 1;
EN = 1;
d_lcd = (d_lcd & 0x0F)|(ascii & 0xF0);
```

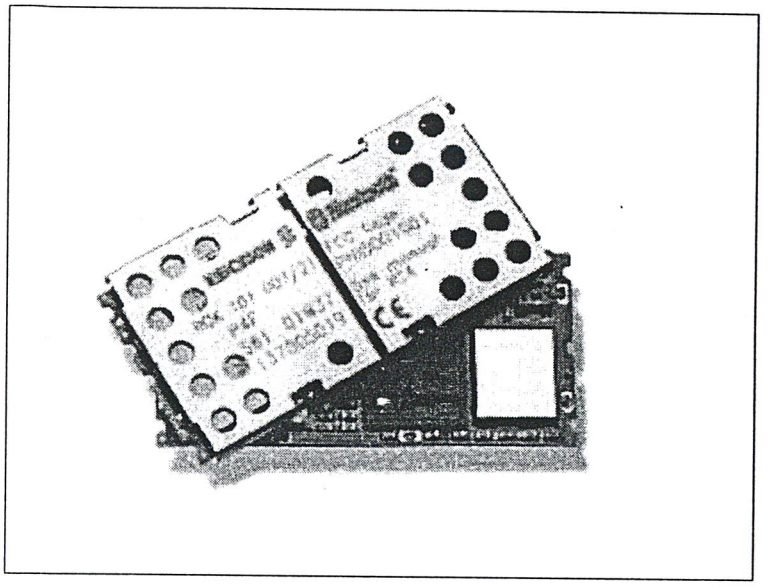
```
lcd_enable();
d_lcd = (d_lcd & 0x0F)|((ascii<<4) & 0xF0);
lcd_enable();
}
void lcd_init(void){
write_instruct(0x33);  //function set DL = 1 8 bit
write_instruct(0x32);  //function set DL = 0 4 bit
write_instruct(0x28);  //function set
write_instruct(0x1c);  //cursor display
write_instruct(0x06);  //entry mode
write_instruct(0x0C);  //display on off
write_instruct(0x01);  //clr
}
void cls(void){
write_instruct(0x01);
}
void line1(void){
write_instruct(0x80);
}
void line2(void){
write_instruct(0xC0);
}
char putchar(char k){
  lcd_putc(k);
  return(k);
}
```

ROK 101 007

Bluetooth™ Multi Chip Module

Key Features

- Pre-qualified Bluetooth 1.1 Module
- Multi Point Operation, 7 slaves
- RF output power class 2
- FCC and ETSI approved
- Multiple interfaces for different applications
- UART for data (HCI interface)
- PCM for voice
- USB for voice and data (HCI interface)
- I²C interface for controlling of external I²C devices
- Internal crystal oscillator
- HCI firmware over USB and UART included



Description

ROK 101 007 is a module for implementing Bluetooth functionality into various electronic devices. The module consists of three major parts; a baseband controller, a flash memory and a radio that operates in the globally available 2.4 – 2.5 GHz free ISM band. Both data and voice transmission is supported by the module. Communication between the module and the host controller is carried out using a full-speed USB interface compliant with USB Specification 2.0 or a UART/PCM interface. When using the USB interface, the module appears as a USB slave device and therefore requires no PC resources. ROK 101 007, which is compliant with Bluetooth version 1.1, is a Class 2 Bluetooth Module (0 dBm) and is type-approved. The module supports all Bluetooth profiles.

Suggested Applications

- Computers and peripherals
- Handheld devices and accessories
- Access points

ERICSSON

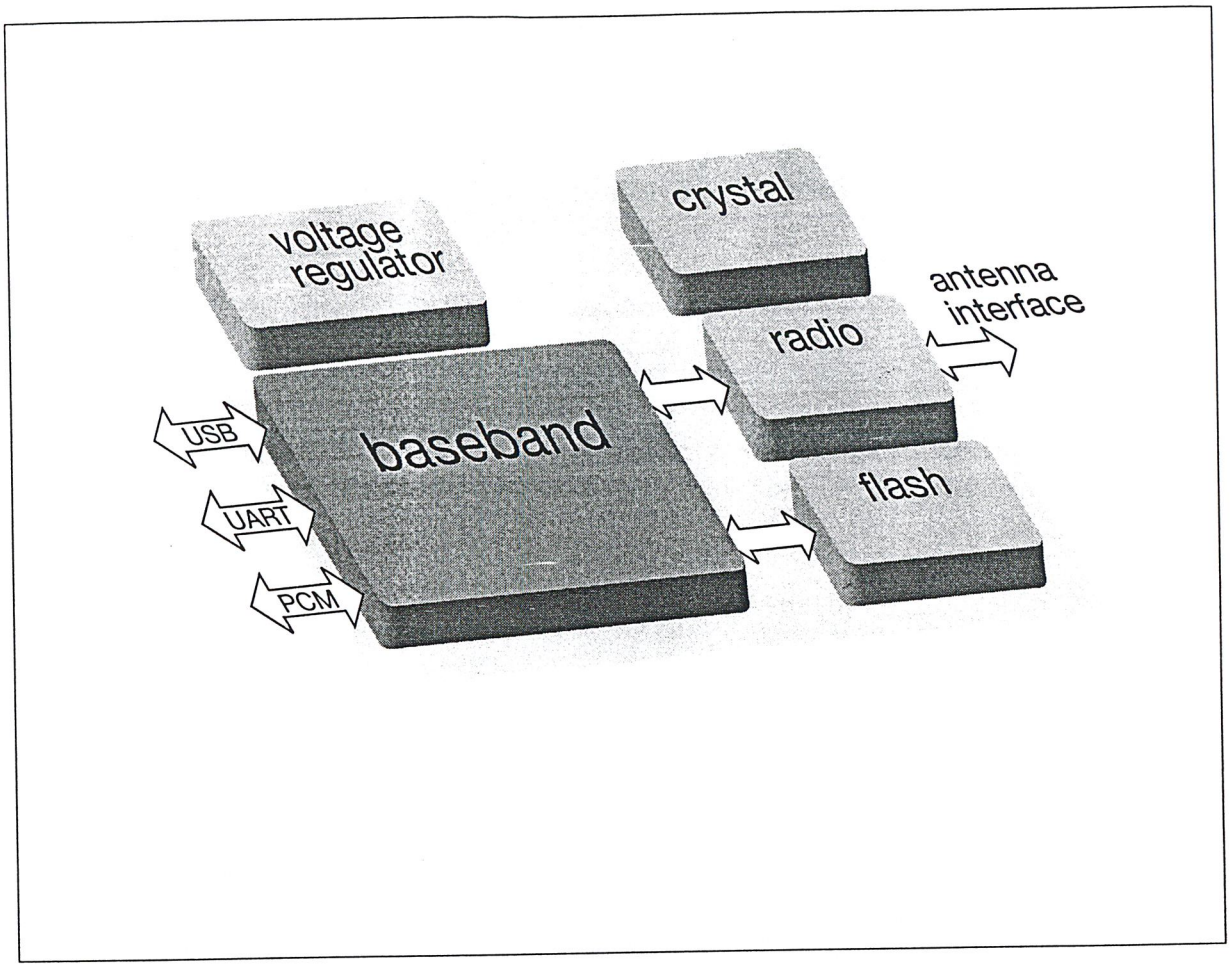


Figure 1. Block Diagram.

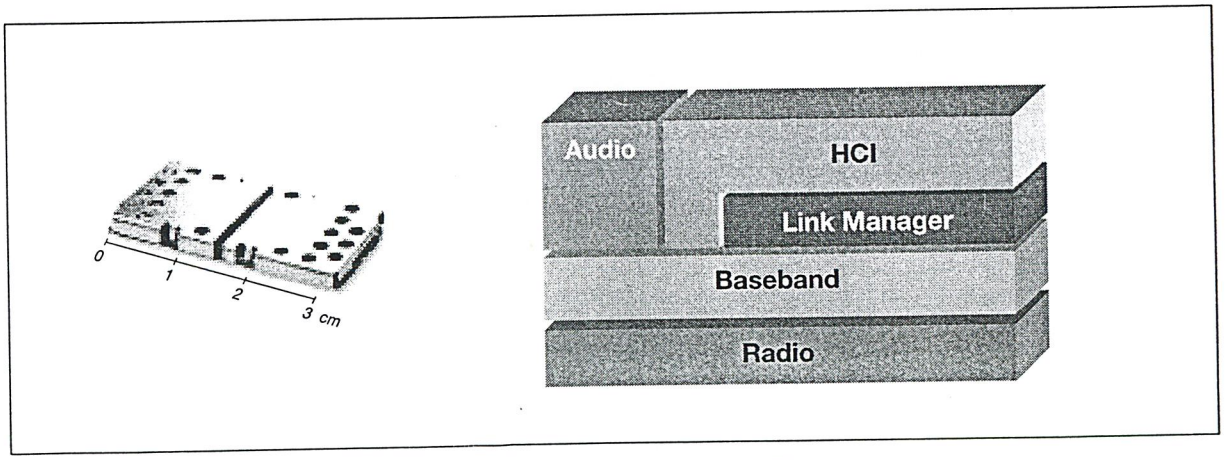


Figure 2. Actual size of the Bluetooth Module and also showing the HW and FW stack.

Absolute Maximum Ratings

Parameter	Symbol	Min	Typ	Max	Unit
Temperature					
Storage temperature	T_{Stg}	-30		+85	°C
Operating temperature	T_{Amb}	0		+75	°C
Power Supply					
V_{CC}	V_{CC}	-0.3		+7.0	V
V_{CC_IO}	V_{CC_IO}	-0.8		+3.6	V
Digital Inputs					
Input low voltage	V_{IL}	-0.5			V
Input high voltage	V_{IH}			V_{CC_IO}+0.3	V
Antenna Port					
Input RF power	In-band Out of band			15 15	dBm dBm

Recommended Operating Conditions

Parameter	Condition	Symbol	Min	Typ	Max	Unit
Temperature						
Ambient temperature, Test		T_{Amb}	0	+23	+75	°C
Power Supply						
Supply Voltage		V_{CC}	3.175	3.3	7.0	V
I/O Ports Supply Voltage		V_{CC_IO}	2.7	3.3	3.6	V

Electrical Characteristics

Unless otherwise noted, the specification applies for T_{Amb} = 0 to +75°C, V_{CC} = 3.3 V and VSWR = 2:1.

DC Specifications

Parameter	Condition	Symbol	Min	Typ	Max	Unit
Digital Inputs						
Logical Input	High Except ON signal	V_{IH1}	0.7xV_{CC_IO}		V_{CC_IO}	V
Logical Input Low	Except ON & RESET# signal	V_{IL1}	0		0.3xV_{CC_IO}	V
Logical Input High	ON signal only	V_{IH2}	2.0		V_{CC}	V
Logical Input Low	ON signal only	V_{IL2}	0		0.4	V
RESET# Input Low	RESET# signal only	V_{RESET}	0		0.4	V
Digital Outputs						
Logical Output High		V_{OH}	0.9xV_{CC_IO}		V_{CC_IO}	V
Logical Output Low		V_{OL}	0		0.1xV_{CC_IO}	V
Average Current Consumption	ICC + ICC_IO					
HW Shutdown state	'ON' is low and 'VCC_IO' is grounded	I_{Shw}		1		µA
Idle state	After HCI – reset	I_{Idle}		3.9	6	mA
Connection	Master mode Slave mode	I_{CM} I_{CS}		35 34	45 45	mA mA
Hold mode		I_{Hold}		32	40	mA
Park mode	Beacon interval 1.28s	I_{Park}		32	40	mA
Sniff mode	Sniff interval 1.28s	I_{Sniff}		32	40	mA
Page Scan	Mandatory page scan mode	I_{PSM}		39	52	mA
Inquiry mode		I_{ISM}		37	52	mA

Parameter	Condition	Symbol	Min	Typ	Max	Unit
Timing Performance						
System start-up time from power on				400		ms
RESET# signal duration	Sink current > 1 mA			1		ms
Firmware timer resolution				6.55		ms
PCM Timing Information						
PCM clock frequency	Master mode Slave mode	f_{PCM_CLK} f_{PCM_CLK}	 128	2000	 2000	kHz kHz
PCM sample rate sync. frequency		f_{PCM_SYNC}		8		kHz
PCM clock high period		t_{CCH}	200			ns
PCM clock low period		t_{CCL}	200			ns
PCM_SYNC (setup) to PCM_CLK (fall)		t_{PSS}	100	1000		ns
PCM_SYNC pulse length		t_{PSH}	200	460		ns
PCM_X in (setup) to PCM_CLK (fall)		t_{DSL}	100			ns
PCM_X in (hold) from PCM_CLK (fall)		t_{DSH}	100			ns
PCM_X in out valid from PCM_CLK (rise)		t_{PDLP}	150			ns

RF Specifications

Parameter	Condition	Symbol	Min	Typ	Max	Unit
General						
Antenna load				50		Ω
Transmitter Performance						
TX power			-6	+2	+4	dBm
Spurious emissions TX mode	30 MHz – 1 GHz 1 GHz – 12.75 GHz 1.8 GHz – 1.9 GHz 5.15 GHz – 5.3 GHz				-36 -30 -47 -47	dBm dBm dBm dBm
Receiver Performance (BER ≤ 0.1%)						
Sensitivity level				-77	-71	dBm
Max input level			0	+13		dBm
Spurious Emissions RX mode	30 MHz – 1 GHz 1 GHz - 12.75 GHz			-74 -60	-57 -47	dBm dBm
Out-of-band blocking	30-1910 MHz 1910-2000 MHz 2000-2399 MHz 2484-3000 MHz 3000-12750 MHz		+4 -10 -27 -27 -10			dBm dBm dBm dBm dBm

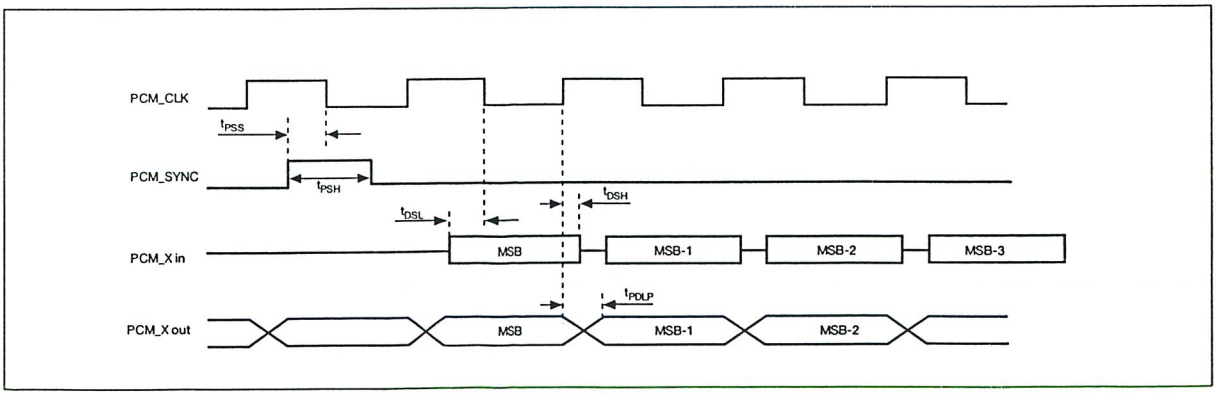


Figure 3. PCM timing.

Pin Description

Pin	Pin Name	Type	Direction	Description
A1	PCM_IN	CMOS	In	PCM data, see note 1
A2	PCM_OUT	CMOS	Out	PCM data, see note 1
A3	PCM_SYNC	CMOS	In/Out	Sets the PCM data sampling rate, see note 1
A4	PCM_CLK	CMOS	In/Out	PCM clock that sets the PCM data rate, see note 1
A5	RXD	CMOS	Input	RX data to the UART
A6	RTS	CMOS	Input	Flow control signal, Request To Send data from UART, see note 1
B1	D+	CMOS	In/Out	USB data pin
B2	D-	CMOS	In/Out	USB data pin
B3	GND	Power	Power	Signal ground
B4	WAKE_UP	CMOS	Output	Indicates that the module wants to be attached to the USB. Active High.
B5	TXD	CMOS	Output	TX data from the UART
B6	CTS	CMOS	Output	Flow control signal, Clear To Send data from UART
C1	DETACH	CMOS	Input	Indicates that the USB host wants to detach the module. Active High.
C2	ON	Power	Input	When tied to VCC, the module is enabled.
C3	I²C_CLK	CMOS	Output	I²C clock signal
C4	VCC_IO	Power	Power	External supply rail to the Input / Output ports
C5	NC	-	-	Do not connect
C6	VCC	Power	Power	Supply Voltage
R1	GND	Power	Power	Signal ground
R2	GND	Power	Power	Signal ground
R3	RESET#	CMOS	Input	Active low reset, see note 2
R4	NC	-	-	Do not connect
R5	NC	-	-	Do not connect
R6	NC	-	-	Do not connect
T1	GND	Power	Power	Signal Ground
T2	ANT	RF	In/Out	50 Ω Antenna connection
T3	GND	Power	Power	Signal Ground
T4	NC	Power	Power	Test point, internal voltage regulator - Do not connect
T5	NC	-	-	Do not connect
T6	I²C_DATA	CMOS	In/Out	I²C data signal

Notes

1. 100 kΩ pull-up resistors to VCC_IO are used on the module. PCM signals direction is programmable.
2. RESET# signal must be fed from an open drain output.

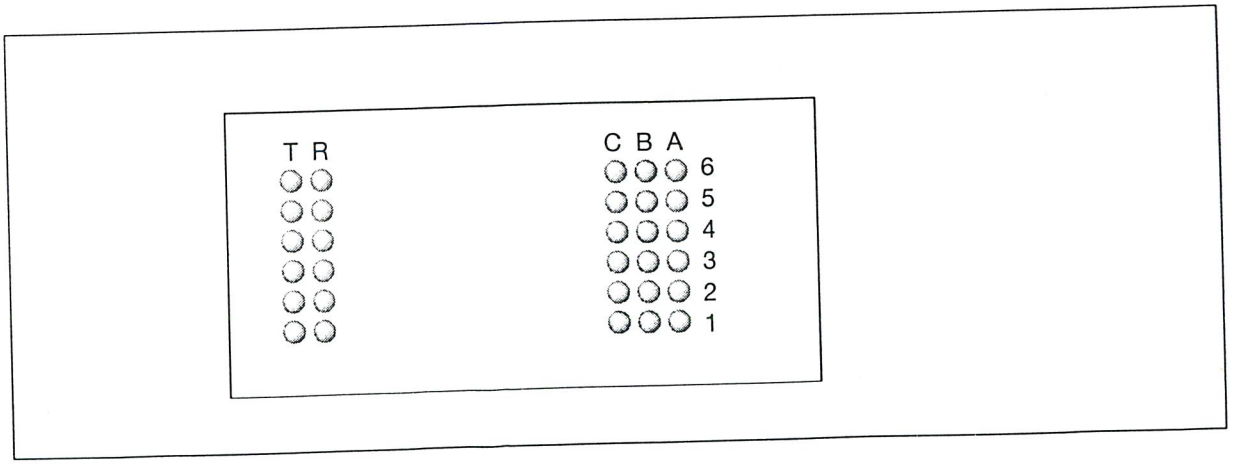


Figure 4. Pinout drawing.

Mechanical Specification

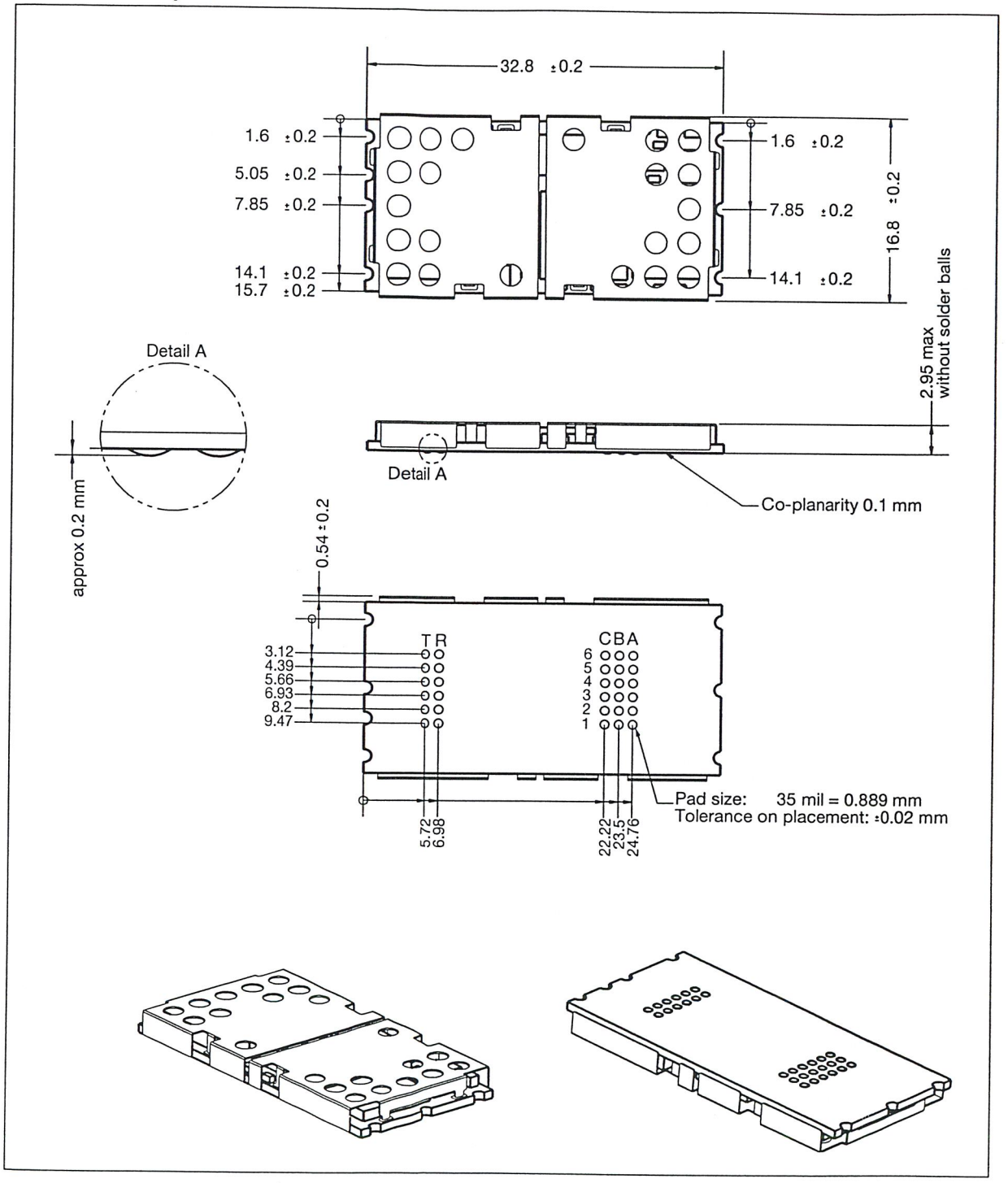


Figure 5. Mechanical dimensions.

Functional Description

The ROK 101 007 is a complete Bluetooth module that has been specified and designed according to the Bluetooth System v1.1. Its implementation is based on a high-performance integrated radio transceiver (PBA 313 01/3), a baseband controller, a flash memory and surrounding secondary components.

ROK 101 007 has five major operational blocks. Figure 6 illustrates the interaction of the various blocks. The functionality of each block is as follows:

1. Radio functionality is achieved by using the Bluetooth Radio, PBA 313 01/3.
2. The baseband controller is an ARM7-Thumb based chip that controls the operation of the radio transceiver via one of the interface methods; USB or UART. Additionally, the baseband controller has a PCM Voice interface and I²C interface.
3. A Flash memory is used together with the baseband controller. Please, refer also to the Firmware section.
4. The power management block regulates and filters the supply voltage.
5. The internal clock frequency is 13 MHz and is generated from a crystal oscillator.

Bluetooth Module stack

The Host Controller Interface (HCI) handles the communication by the transport layer through the UART or USB interface with the host. The Baseband and radio provides a secure and reliable radio link for higher layers. The following sections describe the Bluetooth module stack in more detail, see also figure 7. It is implemented in accordance with and complies with the Specification of the Bluetooth System v1.1.

Bluetooth Radio Interface

The Bluetooth module is a class 2 device with 4 dBm maximum output power with no power control needed. Nominal range of the module with a typical antenna is up to a range of 10 m (at 0 dBm). It is compliant with FCC and ETSI regulations in the ISM band.

Baseband

Bluetooth uses an ad-hoc net structure with a maximum of eight active units in a single piconet. By default the first unit setting up a connection is the master of the link. (See Application Note 1/1522-ROK 101 007 Uen.) The master transmits in the even timeslots and the slave transmits in the odd timeslots. For full duplex transmission, a Time-Division Duplex (TDD) scheme is

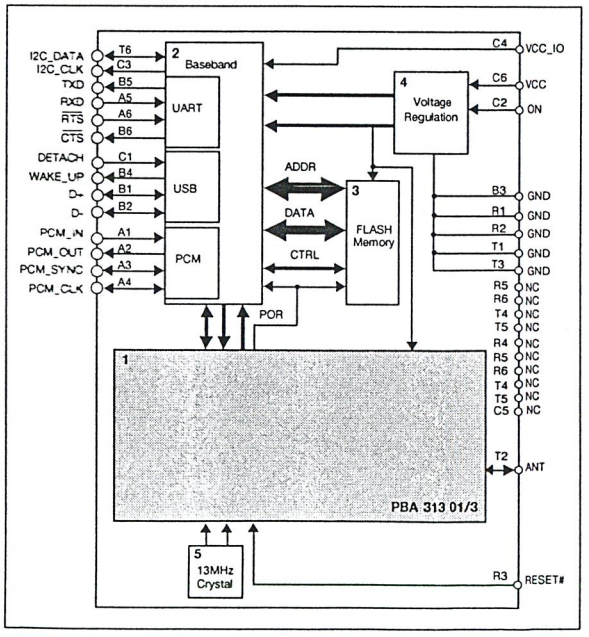


Figure 6. Simplified Block Diagram.

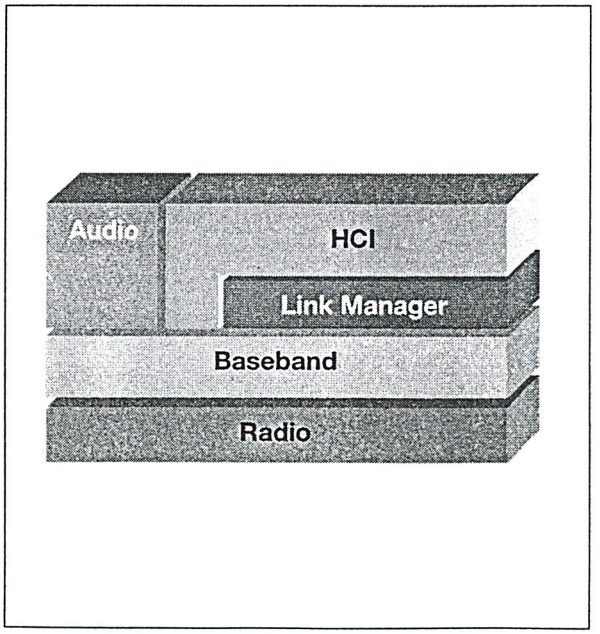


Figure 7. HW/FW parts included in the Ericsson Buetooth module.

used. Packets are sent over the air in timeslots with a nominal length of 625 µs. A packet can be extended to a maximum of 5 timeslots (DM5 and DH5 packets) and is then sent by using the same RF channel for the entire packet. Two types of connections are provided, Asynchronous Connectionless Link (ACL) for data and the Synchronous Connection Oriented Link (SCO) for voice. Three 64 kb/s voice channels can be supported simultaneously. Furthermore, there are also packets used for link control purposes. A variety of different packet types with error correction schemes and data rates can be used over the air interface, see table 1. Also asymmetric communication is available for high-speed communication in one direction. The Baseband provides the link setup and control routines for the layers above. Furthermore, the Baseband also provides Bluetooth security like encryption, authentication and key management. Please refer to the Specification of the Bluetooth System v1.1 part B for in-depth information regarding the Baseband.

Firmware (FW)

The module includes firmware for the host controller interface, HCI, and the link manager, LM. The FW resides in the Flash and is available in object code format. The module supports Device Firmware Upgrade (DFU) over UART or USB. An Application Note, 3/1522-ROK 101 007 Uen, describing the flash procedure is available.

Type	User Payload (bytes)	FEC	CRC	Symmetric Max. rate	Asymmetric Max.rate
ID	na	na	na	na	na
NULL	na	na	na	na	na
POLL	na	na	na	na	na
FHS	18	2/3	yes	na	na

Link control packets

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

ACL packets

Type	Payload header (bytes)	User Payload (bytes)	FEC	CRC	Symmetric Max. rate (kb/s)
HV1	na	10	1/3	no	64.0
HV2	na	20	2/3	no	64.0
HV3	na	30	no	no	64.0
DV	1D	10+(0-9) D	2/3 D	Yes D	64.0+57.6 D

SCO packets

Table 1: Link Control Packets Table, ACL Packets Table, SCO packets.

Link Manager (LM)

The Link Manager in each Bluetooth module can communicate with another Link Manager by using the Link Manager Protocol (LMP) which is a peer to peer protocol, see figure 8. The LMP messages have the highest priority and are used for link-setup, security, control and power saving modes. The receiving Link Manager filters out the message and does not need to acknowledge the message to the transmitting LM due to the reliable link provided by the Baseband and the radio. LM to LM communication can take place without actions taken by the host. Discovery of features at other Bluetooth enabled devices nearby can be found and saved for later use by the host. Please refer to the Specification of the Bluetooth System v1.1 part C for in-depth information regarding the LMP.

Host Control Interface (HCI)

The HCI provides a uniform command I/F to the Baseband and Link Manager and also to HW status registers. The HCI I/F is accessed through UART or USB. There are three different types of HCI packets:

- HCI command packets – from host to Bluetooth module HCI.
- HCI event packets – from Bluetooth module HCI to host.
- HCI data packets – going both ways.

It is not necessary to make use of all different commands and events for an application. If the application is aimed at a pre-specified profile, the capabilities of such a profile is necessary to adjust to – see Specification of the Bluetooth System v1.1 Profiles and Application Note 5/1522-ROK 101 007 Uen.

a) With the HCI UART Transport Layer on top of HCI, the module will communicate with a host through the UART I/F. The PCM I/F is also available for communicating voice.

b) With the HCI USB Transport Layer on top of the HCI, the module will communicate with a host through the USB. Detach and Wake_up signals are also available for notebook implementations. Please refer to the Specification of the Bluetooth System v1.1 part H: 1-4 for in-depth information regarding the HCI and different transport layers.

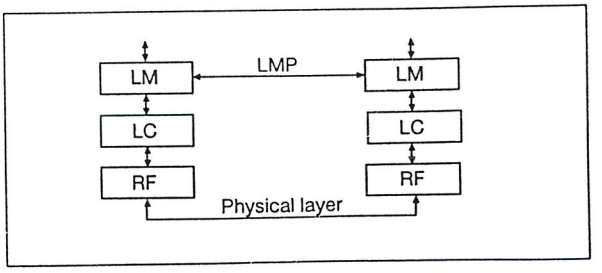


Figure 8. Link manager.

Module HW Interfaces

UART Interface

The UART implemented on the module is an industry standard 16C450 and supports the following baud rates: 300, 600, 900, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400 and 460800 bits/s. See table 1 regarding data rates for different packet types. 128 byte FIFOs are associated with the UART. Four signals will be provided for the UART interface. TxD & RxD are used for data flow and RTS & CTS are used for flow control. Please refer to the Specification of the Bluetooth System v1.1 part H: 4 regarding the HCI and UART transport layers.

USB Interface

The module is a USB full-speed class device (12 Mbps) that has the full functionality of a USB slave and is compliant to the USB 2.0 specification (If VCC_IO > 3.11 V). Data transfer occurs on the bi-directional ports, D+ & D-. Additionally, there are two side band signals for a notebook application. Two side band signals, Wake_up and Detach, are used to control the state from which the notebook resumes. When the host is in a power down mode, Wake_up wakes the host up when the Bluetooth system receives an incoming connection. The host indicates that it is in Suspend mode by using the Detach signal. See Application Note 2/1522-ROK 101 007 Uen.

PCM Voice Interface

The standard PCM interface has a sample rate of 8 kHz (PCM_SYNC). The PCM clock is variable between 200 kHz and 2.0 MHz. The PCM data can be linear PCM (13-16 bit), μ-Law (8 bit) or A-Law (8 bit). The PCM I/F can be either master or slave providing or receiving the PCM_SYNC, see figure 9. Redirection of PCM_OUT and PCM_IN can be accomplished as well. Over the air the encoding is programmable to be CVSD, A-Law or μ-Law. Preferably the robust CVSD encoding should be used.

I²C Interface

A master I²C I/F is available on the module. This is used to control external I²C devices. The control of the I²C pins are performed by Ericsson specific HCI commands available in the FW implementation. See Application Note 4/1522-ROK 101 007 Uen.

Antenna

The ANT pin should be connected to a 50 Ω antenna interface, thereby supporting the best signal strength performance, VSWR should not be higher than 2:1. Ericsson Microelectronics can recommend application specific antennas.

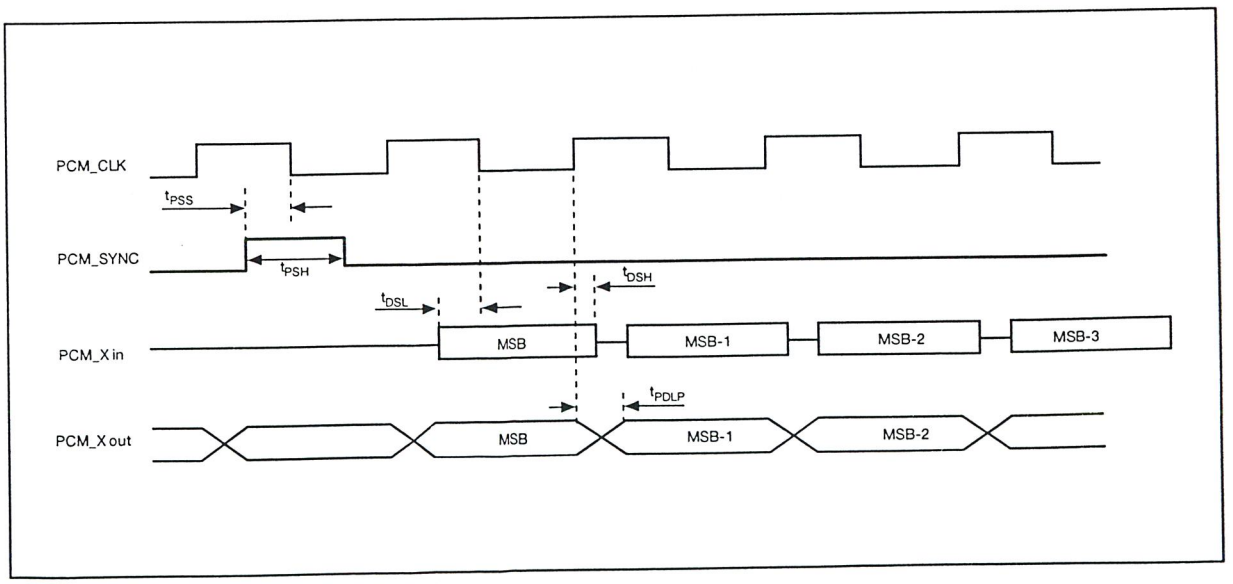


Figure 9. PCM timing.

Design Guidelines

Power-up Sequence

There is no need for a power up sequence if VCC, ON and VCC_IO are tied together. A power up sequence, if used, shall be applied accordingly:
Connection of the supply rails, GND and then VCC; then the ON signal should be applied in order to initiate the internal regulators; and finally the VCC_IO supply rail can be activated, see figure 10 and table 2 for timing information. The power-down sequence is similar to the power-up procedure but in the reverse format. Therefore, the disconnection of the signals shall be as follows: VCC_IO, ON, VCC and finally GND.

RESET#

The assignment of the RESET# input is to generate a reset signal to the module. During power-up the reset signal is set low automatically so that power supply glitches are avoided. Therefore no reset should be required after power-up. When implementing an external RESET#, the signal should be fed from an open drain output.

Power

There are three inputs to the Voltage Management section (VCC, VCC_IO, ON). VCC is the supply voltage that is typically 3.3 V. A separate power supply rail (VCC_IO) is provided for the I/O ports, UART, PCM and USB. VCC_IO can either be connected to VCC or to a dedicated supply rail, which is the same as the logical interface of the host. The ON signal is controlling the internal regulators on or off.

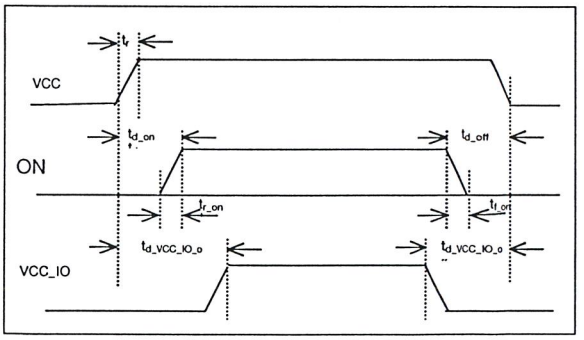


Figure 10. Power up sequence.

Parameter	Min	Nom	Max	Unit
t_r			1	ms
t_{r_on}			60	µs
t_{f_on}			60	µs
t_{d_on}	t_r			µs
t_{d_off}				µs
$t_{dVCC\_IO\_on}$	t_{d_on}			µs
$t_{dVCC\_IO\_off}$	t_{d_off}			µs

Table 2. Power up parameters.

Antenna

It is very important to keep the antenna output routing at 50 Ω (VSWR ≤ 2:1) all the way to the antenna in order to maintain the radio performance listed in this data sheet and thereby the FCC and ETSI approvals. For the routing underneath the module, the modules ground plane should be considered.

Shielding / EMC Requirements

The module has its own RF shielding and is approved according to the standards by FCC and ETSI. If the approval number is not visible on the outside when the module is utilized in the final product, an exterior label must state that there is a transmitter module inside the product.

Ground

Ground should be distributed with very low impedance as a ground plane. Connect all GND pins to the ground plane.

Assembly Guidelines

Solder Paste

The ROK 101 007 module is made for surface mounting and the SSP connection pads have been formed after printing eutectic Tin/Lead solder paste. To assembly the module, flux must be applied at the target surface. This can either be done by fluxing the area (this is only preferable if fluxing is a part of the assembly process) or by just printing solder paste on the pads (flux is included in the paste). A preferred solder paste height is 127µm (5 mil).

Soldering Profile

It must be noted that the module should not be allowed to be hanging upside down in the reflow operation. This means that the module has to be assembled on the side of the PCB that is soldered last. The reflow process should be a regular surface mount soldering profile (full convection strongly preferred); the ramp-up should not be higher than 3 °C/s and with a peak temperature of 210-225 °C during 10-20 seconds. Max sloping rate should not be higher than 4 °C/s (see example of reflow profile in figure 11).

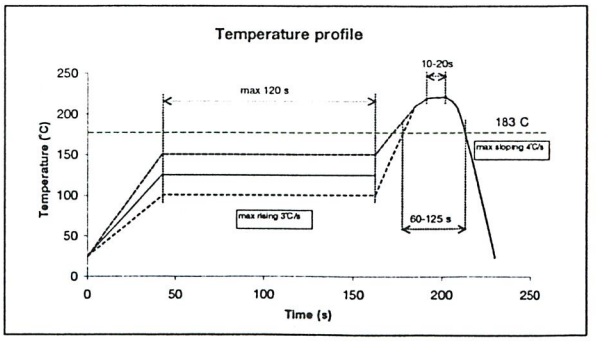


Figure 11. Eutectic SnPb-solder profile.

Pad Size

It is recommended that the pads on the PCB should have a diameter of 0.7-0.9 mm. The surface finish on the PCB pads should be Nickel/Gold or a flat Tin/Lead surface or OSP (Organic Surface Protection).

Placement

The placement machine should be able to recognize odd LGA combinations (all ball recognition preferred) and be able to pick the component asymmetrically. The module contains a flat pick-area of 10 mm diameter minimum. The centre of gravity of the module is situated a few mm to the 'left' from the centre of the module - towards the large Bluetooth radio. It is thus recommended to change the offset in the assembly unit such that the nozzle picks the module just 'left' from the gap. Experience shows that this will result in proper assembly. The weight of the module is typically 2.8 g.

Storage

Keep the component in its dry pack when not yet using the reel. After removal from the dry pack ensure that the modules are soldered onto the PCB within 48 hours.

Module marking

Each module is marked on the shield with the following information:

1. Ericsson logotype
2. Product No with index
3. Revision state
4. Manufacturing unit code
5. Production year and week
6. Serial number
7. Bluetooth trademark
8. FCC product code
9. See manual for RTA approval
10. CE marking

Reel marking

The reel, reel box and dry pack has a label with the following information:

1. Ericsson product number with revision
2. Customer product number with revision
3. Quantity
4. Reel-ID. (Batch No)
5. Factory code
6. Manufacturing date
7. Country of origin
8. Ericsson logotype

1-6 above is also printed in BAR-code format

Ordering Information

Package Part No.
30 SSP ROK 101 007/2

Packaging

The modules will be delivered in a tape & reel and dry pack, protecting them from ESD and mechanical shock. The tape width is 44 mm and the pitch is 24 mm. The diameter of the reel is 13 inches and it contains 500 modules.

Abbreviations

ASIC	-	Application Specific Integrated Circuit
BER	-	Bit Error Rate
CMOS	-	Complementary Metal Oxide Semiconductor
DCE	-	Data Circuit terminating Equipment
HCI	-	Host Controller Interface
ISM	-	Industrial Scientific and Medical
PCB	-	Printed Circuit Board
PCM	-	Pulse Code Modulation
RX	-	Receive
SIG	-	Special Interest Group
SSP	-	Screen Solder Print
TX	-	Transmit
UART	-	Universal Asynchronous Receiver Transmitter
USB	-	Universal Serial Bus