ชุดนวดระบบสั่นสะเทือนควบคุมการทำงานด้วยไมโครคอนโทรลเลอร์
VIBRATION BODY MASSAGE SUIT CONTROLLED BY MICROCONTROLLER

นายดุลยวัฒน์ แสงนภาฉาย
Dunyawat Sangnapachai
นายประพันธ์ แก้วมหานิล
Prapan Kaewmahanil

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2560

ชุดนวดระบบสั่นสะเทือนควบคุมการทำงานด้วยไมโครคอนโทรลเลอร์

VIBRATION BODY MASSAGE SUIT CONTROLLED BY MICROCONTROLLER

โดย

นายดุลยวัฒน์ แสงนภาฉาย

นายประพันธ์ แก้วมหานิล

อาจารย์ที่ปรึกษา

ผศ.ดร. แสงระวี บัวแก้ว

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2560

ปริญญานิพนธ์	ปีการศึกษา 2560
สาขาวิชา	วิศวกรรมอิเล็กทรอนิกส์
คณะ	วิศวกรรมศาสตร์
	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
เรื่อง	ชุดนวดระบบสั่นสะเทือนควบคุมการทำงานด้วยไมโครคอนโทรลเลอร์
	Vibration Body Massage Suit Controlled by Microcontroller
ผู้จัดทำ	นาย ดุลยวัตน์ แสงนภาฉาย รหัสประจำตัว 57010484
	นาย ประพันธ์ แก้วมหานิล รหัสประจำตัว 57010738

ปริญญานิพนธ์นี้ผ่านการตรวจสอบโดยอาจารย์ที่ปรึกษาแล้ว

(ผศ.ดร.แสงระวี บัวแก้ว)
อาจารย์ที่ปรึกษา

ชุดนวดระบบสั่นสะเทือนควบคุมการทำงานด้วยไมโครคอนโทรลเลอร์

นายดุลยวัฒน์ แสงนภาฉาย, นายประพันธ์ แก้วมหานิล

อาจารย์ที่ปรึกษา : ผศ.ดร. แสงระวี บัวแก้ว

ภาควิชาวิศวกรรมอิเล็คทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

บทคัดย่อ

ปริญญานิพนธ์นี้นำเสนอชุดนวดระบบสั่นสะเทือนควบคุมการทำงานด้วยไมโครคอนโทรลเลอร์ ชุดนวดระบบสั่นสะเทือนนี้ทำงานภายใต้การควบคุมของไมโครคอนโทรลเลอร์เบอร์ 16F887 การทำงานของชุดนวดระบบสั่นนี้จะถูกควบคุมโดยโปรแกรมภาษาซีที่ได้มีการตั้งเงื่อนไขและรีจิสเตอร์ ซึ่งไมโครคอนโทรลเลอร์จะรับค่าสองทางแบ่งเป็นจากรีโมทซึ่งเป็นสัญญาณอนาลอกและจากสัญญาณบลูทูธที่ได้มีการพัฒนาแอพพลิเคชั่นในระบบปฏิบัติการแอนดรอยด์บนสมาร์ทโฟนเพื่อให้เกิดความคล่องตัวในการควบคุมชุดนวด ซึ่งเมื่อไมโครคอนโทรลเลอร์ได้รับคำสั่งที่ส่งมาแล้ว ไมโครคอลโทรลเลอร์จะกำเนิดสัญญาณพัลส์ส่งไปยังมอเตอร์ไฟฟ้ากระแสตรง 12 ลูก ซึ่งอยู่ในชุดนวดที่มีความถี่และความเร็วรอบที่ต่างกันและสามารถควบคุมได้ด้วยสมาร์ทโฟน เมื่อมอเตอร์ไฟฟ้ากระแสตรงหมุนจะเกิดแรงสั่นสะเทือนส่งผ่านจากชุดนวดไปยังส่วนต่างๆของร่างกายของผู้สวมใส่ชุดนวดนี้

คำสำคัญ – สั่นสะเทือน; ไมโครคอนโทรลเลอร์ ; บูลทูธ ; สัญญาณพัลส์

VIBRATION BODY MASSAGE SUIT CONTROLLED BY MICROCONTROLLER

Mr.Dunyawat Sangnapachai, Mr. Prapan Kaewmahanil

Advisor: Asst.Prof.Dr. Sangrawee Buakaew

Department of Electronics Engineering

Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand

ABSTRACT

This thesis presents a vibration body massage suit controlled by microcontroller. The massage suit can generate vibration which controlled by microcontroller 16F887. The microcontroller is programmed to receive two different sources of data which are analog signals from remote controller. For controlling the suit conveniently, android software application is developed on smartphone. After the microcontroller received the data, it will generate PWM signal to twelve DC motors inside the suit and generate vibration by spinning with different frequency and speed (RPM). This vibration is to send through the whole body of a person who wear the suit.

Keywords — Vibration ; Microcontroller ; Bluetooth ; PWM signal

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้ประสบผลสำเร็จไปได้ด้วยดีทั้งนี้เนื่องจากคำแนะนำของอาจารย์ที่ปรึก ผศ.ดร. แสงระวี บัวแก้ว อาจารย์ที่ปรึกษา และ อาจารย์ท่านอื่นๆในภาควิชาวิศวกรรมอิเล็กทรอนิกส์ สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์ ผู้จัดทำขอกราบขอบพระคุณในความอนุเคราะห์จากอาจารย์ทุกท่านช่วยเหลือในการทำโครงการนี้ และขอขอบคุณเพื่อนๆทุกคนที่มาช่วยเหลือและแนะนำในการทำโครงการนี้ จึงทำให้โครงการนี้สำเร็จไปได้ด้วยดี และคณะผู้จัดทำได้รับความรู้เพิ่มมากยิ่งขึ้น ดังนั้นคณะผู้จัดทำขอขอบพระคุณทุกๆท่าน มา ณ ที่นี้

คณะผู้จัดทำ

ดุลยวัฒน์ แสงนภาฉาย

ประพันธ์ แก้วมหานิล

สารบัญ

รายการ **หน้า**

สารบัญรูป

รูปที่ **หน้า**

รูปที่ **หน้า**

สารบัญตาราง

ตารางที่ **หน้า**

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของโครงงาน

เนื่องด้วยทางภาควิชาวิศวกรรมอิเล็กทรอนิกส์ มีนโยบายให้นักศึกษาทำโครงงานในวิชา โปรเจค เพื่อพัฒนาทักษะในการปฏิบัติงานตามความสนใจของนักศึกษาและเพื่อต่อยอดความรู้ไปใช้ในการทำงานในอนาคต ในปัจจุบันพบว่า มีผู้มีปัญหาเกี่ยวกับการปวดเมื่อยตามร่างกาย อันเนื่องมาจากการทำงานหนักการอยู่ในอิริยาบทเดิมเป็นระยะเวลานาน และผู้ป่วยที่เป็นอัมพาตเนื่องจากมีการอ่อนแรงของกล้ามเนื้อทำให้ไม่สามารถขยับได้ จากการศึกษาในด้านแรงสั่นสะเทือนที่ส่งผลต่อกล้ามเนื้อนั้นพบว่าแรงสั่นสะเทือนจะช่วยคลายกล้ามเนื้อที่ตึง การสั่นสะเทือนสามารถลดความรัดกุมโดยการเพิ่มการไหลเวียนโลหิตและอุณหภูมิของกล้ามเนื้อการไหลเวียนของเลือดเพิ่มขึ้นทำให้อุณหภูมิของกล้ามเนื้อเพิ่มขึ้นจึงทำให้เกิดการผ่อนคลาย เมื่อกล้ามเนื้อผ่อนคลายผ่อนคลายจะรู้สึกผ่อนคลาย จึงได้มีการคิดค้นอุปกรณ์ที่ช่วยในการบรรเทาอาการปวดเมื่อยตามร่างกายและช่วยกระตุ้นการทำงานของกล้ามเนื้อหรือเรียกสั้นๆว่า “ชุดนวดระบบสั่นสะเทือน” โดยวิธีการส่งแรงสั่นสะเทือนไปที่กล้ามเนื้อส่วนต่างๆที่อ่อนแรงให้การทำงานบริเวณนั้นๆให้ดีขึ้น

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

1. เป็นการฝึกปฏิบัติงานกันเป็นหมู่คณะ
2. ตอบสนองต่อนโยบายของภาควิชาวิศวกรรมอิเล็กทรอนิกส์ เพื่อพัฒนาผลงานที่จะเป็นประโยชน์ ต่อสังคมต่อไป
3. เพื่อศึกษาในการเขียนภาษาซี ซึ่งสามารถนำไปใช้งานได้โดยทั่วไป
4. เพื่อศึกษาการออกแบบวงจรอิเล็กทรอนิกส์
5. เพื่อศึกษาในการเขียนภาษาจาวา ซึ่งใช้ในการสร้าง Application
6. เพื่อสามารถบูรณาการความรู้ที่ได้รับทางทฤษฎีออกมาเป็นผลงานที่ใช้ได้จริง
7. เพื่อนำไปต่อยอดและสามารถนำออกสู่ตลาดได้

1.3 สมมุติฐานการศึกษา

สามารถเข้าใจถึงทฤษฎี รวมถึงการทำงานของ Microcontroller ที่ใช้ในการควบคุม เช่น ฟังก์ชั่นการทำงาน การตั้งค่ารีจิสเตอร์ต่างๆ ของพอร์ทที่ต้องการใช้งาน อีกทั้งเข้าใจถึงทฤษฎีของภาษาซีและภาษาจาวา ซึ่งนำไปใช้ในการออกแบบ Application เพื่อนำไปใช้งาน

1.4 ขอบเขตของการศึกษา

1. ทำการออกแบบวงจรควบคุมของ Microcontroller เพื่อใช้ในการควบคุมการทำงานของมอเตอร์
2. สามารถเขียนโปรแกรม MPLAB IDE ใช้งานควบคู่กับ Microcontroller เบอร์ PIC16F887
3. สามารถเขียนโปรแกรมควบคุมการทำงานของชุดนวดโดยใช้ Smartphone ระบบ Android

1.5 ระยะเวลาในการดำเนินโครงงาน

แผนการดำเนินงาน	เดือน	มกราคม				กุมภาพันธ์				มีนาคม				เมษายน			
	สัปดาห์	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1). ออกแบบชุดนวดที่ใช้ใส่ร่างกาย		★	★														
2). ออกแบบลายวงจร			★	★	★												
3). ออกแบบแอพพลิเคชั่น						★	★	★									
4). เขียนโปรแกรมภาษาซี									★	★	★	★					
5). ตัดเย็บชุดนวด									★	★							
5). ทดสอบการทำงานทั้งหมด													★	★			
6). จัดทำรายงานและ Presentation														★	★	★	

บทที่ 2
หลักการและทฤษฎี

2.1) ประวัติความเป็นมาของการใช้แรงสั่นสะเทือนในการบำบัด

ในช่วงประมาณปี ค.ศ.1850 Dr. Jonas Gustav Vilhelm Zander หรือเรียกกันโดยทั่วไปว่า Dr. Gustav Zander นักฟิสิกส์และผู้เชี่ยวชาญด้านศัลยกรรมกระดูกชาวสวีเดน ได้ประดิษฐ์อุปกรณ์ซึ่งใช้สปริง ตุ้มน้ำหนัก และรอก ในการประดิษฐ์อุปกรณ์มากกว่า 70 ชิ้น ซึ่งใช้ในการกายภาพบำบัดและออกกำลังกาย อุปกรณ์ส่วนใหญ่ของ Dr. Gustav Zander นั้นจะใช้หลักการแรงสั่นสะเทือนเป็นหลัก ต่อมาในปี ค.ศ.1895 Dr. John Harvey Kellogg นักฟิสิกส์และนักโภชนาการชาวอเมริกา ได้นำการใช้งานเทคโนโลยีการสั่นสะเทือนเพื่อสุขภาพมาประดิษฐ์เก้าอี้สั่นสะเทือนที่สามารถกระตุ้นการบีบตัวของลำไส้ ซึ่งแก้อาการท้องผูก รักษาอาการปวดหัวและหลัง อีกทั้งยังเพิ่มปริมาณออกซิเจนเข้าสู่ร่างกาย จากนั้นจากโครงการอวกาศรัสเซียได้ใช้เทคโนโลยีการสั่นเทือนมาใช้ป้องกันอาการกระดูกพรุนของนักบินอวกาศตั้งแต่อยู่บนโลกจนกระทั่งช่วงปฏิบัติงานในอวกาศใยสภาพไร้น้ำหนัก ซึ่งปรากฏว่า นักบินอวกาศหลังจากกลับมาจากปฏิบัติภารกิจยังคงมีมวลกระดูกคงเดิมเหมือนก่อนการออกไปปฏิบัติงาน อันเป็นผลมาจากการใช้เครื่องนวดระบบสั่นสะเทือน นอกจากนี้มหาวิทยาลัยในเยอรมันยังศึกษาผลที่ได้จากการใช้แรงสั่นสะเทือนกับร่างกาย เช่น เพิ่มมวลกระดูก เพิ่มมวลกล้ามเนื้อ ปรับสมดุลการไหลเวียนเลือด ฟื้นฟูอาการบาดเจ็บ ลดน้ำหนัก และเผาผลาญไขมัน ทำให้การใช้เทคโนโลยีการสั่นสะเทือนแพร่หลายออกไปยังที่ต่างๆ ทั่วโลก ไม่ว่าจะเป็น เอเชีย ยุโรป อเมริกา ซึ่งในปัจจุบันมีการงานวิจัย สิ่งประดิษฐ์ อย่างหลากหลาย ไม่ว่าจะเป็นการใช้ในรักษาผู้ป่วยในโรงพยาบาล ในด้านฟื้นฟูสมรรถภาพนักกีฬา เครื่องใช้ไฟฟ้าที่ทุกคนสามารถใช้งานได้อย่างสะดวก

2.2) อาการปวดเมื่อยกล้ามเนื้อและกระดูก

อาการปวดเมื่อยกล้ามเนื้อและกระดูกตามส่วนต่างๆ ของร่างกาย โดยส่วนมากพบบ่อยในบริเวณหลัง ไหล่ เอว และต้นคอ โดยสาเหตุนั้นมาจากพฤติกรรมการใช้ชีวิตประจำวันหรือเป็นอาการแทรกซ้อนของโรคที่เป็นอยู่

2.2.1) สาเหตุและอาการของอาการปวดเมื่อย

- Myofascial pain syndrome (MPS) หรือ กลุ่มอาการปวดกล้ามเนื้อและเนื้อเยื่อพังผืด (กลุ่มอาการเอมพีเอส) เป็นสาเหตุที่พบบ่อยมาก และพบร่วมกับกลุ่มอาการปวดอื่นๆได้บ่อย โดยมีอาการปวดตื้อๆ ลึกๆ ปวดร้าวไปบริเวณข้างเคียง บางครั้งปวดพอรำคาญ บางครั้งปวดรุนแรงมากจนเคลื่อนไหวลำบาก มีจุดกดเจ็บหรือจุดที่ไวต่อการกระตุ้น (Trigger point) อยู่ในกล้าม เนื้อหรือในเนื้อเยื่อพังผืด ปวดมากหลังตื่นนอนตอนเช้า หรือหลังใช้กล้ามเนื้อนั้นเป็นระยะเวลา นานอย่างต่อเนื่อง ผู้ชายมีโอกาสเป็นโรคนี้มากกว่าผู้หญิง พบในวัยทำงานมากกว่าวัยอื่นๆ พบบ่อยในกลุ่มพนักงานสำนักงาน (Office) กลุ่มผู้ใช้แรงงาน โดยอาการจะเป็นมากขึ้นถ้ามีการใช้งานกล้าม เนื้อหนักอย่างต่อเนื่องเป็นระยะเวลานาน และใช้งานในท่าที่ไม่เหมาะสม

สาเหตุของการเกิดโรคนี้ เกิดจากการที่กล้ามเนื้อนั้นมีการทำงานหนักอย่างต่อเนื่อง เกิดการสะสมของของเสียในกล้ามเนื้อ ส่งผลให้กล้ามเนื้อมีการหดตัวและขาดออกซิเจน จึงส่งผลให้เกิดอาการปวดกล้ามเนื้อ โดยอาการอาจไม่รุนแรง แค่พอรำคาญ หรือรุนแรงจนส่งผลต่อการเคลื่อนไหว

- โรคไฟโบรมัยอัลเจีย (Fibromyalgia) หรือกลุ่มอาการปวดกล้ามเนื้อ เส้นเอ็นและเนื้อเยื่ออ่อนทั่วร่างกาย ลักษณะสำคัญ คือ อาการปวดจะรุนแรงเมื่อกระตุ้นโดยสิ่งเร้า ซึ่งในคนปกติจะไม่ปวด (Allodynia) เช่น การเปลี่ยนแปลงอุณหภูมิสิ่งแวดล้อม เป็นต้น หรือ อาการปวดทั้งตัว คล้ายกล้ามเนื้อถูกดึงหรือตึงเหมือนทำงานอย่างหนัก ปวดทุกกล้ามเนื้อ คล้ายหมดแรง นอนหลับตื่นขึ้นมาไม่สดชื่น ปวดตึงพังผืดตามข้อต่อต่างๆในช่วงเช้า พบโรคนี้ในผู้หญิงมากกว่าในผู้ชาย พบบ่อยในวัยกลางคนและผู้สูงอายุ โดยสาเหตุยังไม่ทราบแน่นอน สันนิษฐานว่า เกิดจากปัจจัยร่วมทาง จิตใจ ร่างกาย และสิ่งแวดล้อม เพราะผู้ป่วยบางรายจะมีอาการหลังจากมีเหตุการณ์กระตุ้นให้เกิดอาการ เช่น ติดเชื้อไวรัส แบคทีเรีย หรือหลังไม่สบายจากเหตุอื่นๆ ร่วมกับผู้ป่วยส่วนหนึ่งมีภาวะทางจิตใจผิดปกติ เช่น วิตกกังวล ซึม เศร้า มีความผิดปกติของการนอน (นอนไม่หลับ) มีการรับรู้ความเจ็บปวดที่ผิดไป คือ มีความรู้สึกไวเกิน (Hyperesthesia) เช่น ต่อการสัมผัสเบาๆ มีความเครียด การพักผ่อนไม่พอ นอนไม่หลับ อดนอน และการเปลี่ยนแปลงของสิ่งแวดล้อมอย่างรวดเร็ว เช่น อุณหภูมิ จะกระตุ้นทำให้ผู้ป่วยมีอาการรุนแรงขึ้น

- กลุ่มอาการความล้าเรื้อรัง (Chronic fatigue syndrome) หรืออาการอ่อนเพลีย อ่อนล้าของร่างกายเรื้อรัง ลักษณะสำคัญคือ ผู้ป่วยมีอาการอ่อนเพลีย ไม่มีแรง เหมือนมีไข้ต่ำๆ มักมีอาการหลังจากทำงานหนักต่อเนื่อง และพักผ่อนไม่พอ หรือมีโรคทางกายอื่นๆร่วมด้วย เช่น โรคหัวใจ โรคเบาหวาน โรคต่อมไทรอยด์ หรือโรคมะเร็ง อาการอ่อนเพลียจะมากขึ้นถ้าโรคที่ผู้ป่วยเป็นนั้นมีอาการรุนแรงขึ้น หรือไม่สามารถควบคุมอาการได้ ร่วมทั้งมีความเครียด วิตกกังวล นอนไม่หลับ พักผ่อนไม่พอเป็นต้น กลุ่มอาการนี้พบในผู้หญิงบ่อยกว่าในผู้ชาย มักพบในช่วงวัยหนุ่มสาวจนถึงวัยกลางคน

- Polymyalgia rheumatic (PMR) หรือ โรค/ภาวะอักเสบเรื้อรังของกล้ามเนื้อและเนื้อเยื่อพังผืดทั่วร่างกาย มีลักษณะสำคัญ คือ อาการปวดกล้ามเนื้อคล้ายกล้ามเนื้อหดตึงบริเวณไหล่ สะโพก และกล้ามเนื้อคอ และอาจปวดร้าวไปที่ข้อศอก ข้อเข่า และกล้ามเนื้อยึดข้อคล้ายยึดติดช่วงเช้าหลังตื่นนอน ร่วมกับอาการอ่อนเพลีย ไข้ต่ำๆ เบื่ออาหาร น้ำหนักลด
ภาวะนี้พบบ่อยในผู้สูงอายุ ผู้มีความเครียด ความกังวล การพักผ่อนไม่เพียงพอ และอาจมีภาวะติดเชื้อบางชนิดเป็นตัวกระตุ้นให้มีอาการได้ง่ายและรุนแรงขึ้น กลไกการเกิดโรคนี้ที่แน่ชัดยังไม่ทราบ สันนิษฐานว่าอาจมีปัจจัยทางด้านพันธุกรรมและสิ่งแวดล้อมเข้ามาเกี่ยวข้อง โดยพบผู้ป่วยมีความสัมพันธ์กับการตรวจพบยีน/จีน ผิดปกติที่เรียก ว่า HLA-DRB1

โรคนี้พบบ่อยในบางประเทศและบางภูมิภาค เช่น สแกนดิเนเวีย (Scandinavia) พบบ่อยในฤดูร้อนมากกว่าฤดูหนาวและอาจพบร่วมกับการติดเชื้อแบคทีเรียบางชนิดที่ทำให้เกิดการติดเชื้อระบบทางเดินหายใจ คือ Mycoplasma pneumonia และ Chlamydia pneumonia และไวรัสบางชนิดที่ทำให้เกิดโรคไข้ขึ้นผื่น คือ Parvovirus B19 บางการศึกษาพบมีความผิดปกติของระบบต่อมไร้ท่อร่วมด้วย เช่น มีความผิดปกติของฮอร์โมนคอร์ติซอล (Cortisol ฮอร์โมนเกี่ยวกับความเครียดของร่างกาย) และฮอร์โมนอีกหลายชนิด

- Muscle weakness หรือ กล้ามเนื้ออ่อนแรง ลักษณะสำคัญคือ ผู้ป่วยจะมีอาการอ่อนแรงของกล้ามเนื้อส่วนต้นแขน ต้นขา ถ้ามีอาการอ่อนแรงไม่มาก จะรู้สึกเมื่อยล้า และอาจมีอา การปวดกล้ามเนื้อร่วมด้วย กรณีที่มีการอักเสบของกล้ามเนื้อร่วมด้วย เช่น โรคกล้ามเนื้ออักเสบ (Polymyositis) โรคกล้ามเนื้ออ่อนแรงเอมจี (Myasthenia gravis) หรือ โรคกล้ามเนื้ออ่อนแรงเหตุเกลือแร่โปตัสเซียม (Potassium) ต่ำที่เรียกว่า ภาวะ Hypokalemic periodic paralysis ทั้งนี้ผู้ที่มีโอกาสมีกล้ามเนื้ออ่อนแรง จะขึ้นกับแต่ละโรคนั้นๆที่ผู้ป่วยเป็น

2.3) ประโยชน์ของการบำบัดด้วยการสั่นสะเทือน

การนวดแบบสั่นสะเทือนสามารถทำได้โดยนักบำบัดโรคหรือด้วยอุปกรณ์อิเล็กทรอนิกส์ ซึ่งเป็นการใช้แรงในระดับต่างๆ โดยการกดหรือสั่น ลงไปบนส่วนต่างๆ ของร่างกายและกล้ามเนื้อ การสั่นสะเทือนสามารถมีระดับความแรงเพิ่มมากขึ้น โดยการใช้อุปกรณ์อิเล็กทรอนิกส์ที่ออกแบบมาเฉพาะเพื่อวัตถุประสงค์นี้

2.3.1) ช่วยในการผ่อนคลาย การสั่นสะเทือนเป็นเทคนิคที่ช่วยในการผ่อนคลายกล้ามเนื้อที่เกิดการตึง และเพิ่มอุณหภูมิของกล้ามเนื้อ ส่งผลให้การไหลเวียนของเลือดเข้าสู่กล้ามเนื้อมากขึ้น เมื่อกล้ามเนื้อมีอุณหภูมิสูงขึ้นจะส่งผลให้เส้นใยกล้ามเนื้อในส่วนที่ถูกนวดเกิดการผ่อนคลายและคลายตัว ซึ่งช่วยในการเคลื่อนไหวร่างกายได้มากขึ้น

2.3.2) ช่วยเพิ่มความแข็งแรงและมวลของกล้ามเนื้อ การนวดแบบสั่นสะเทือนส่งผลต่อกล้ามเนื้อในส่วนต่างๆของร่างกายให้มีการเคลื่อนไหวเป็นระยะเวลาหนึ่ง ซึ่งส่งผลให้กล้ามเนื้อมีการตื่นตัวและเกิดการเกร็งตัว โดยการเกร็งตัวของกล้ามเนื้อในช่วงหนึ่งจะสามารถเพิ่มความแข็งแรงและความทนทานของกล้ามเนื้อได้

2.3.3) ช่วยเพิ่มภูมิคุ้มกันและการไหลเวียนโลหิต ยิ่งกล้ามเนื้อมีการเกร็งตัวมากขึ้น จะส่งผลให้การไหลเวียนโลหิตในร่างกายเพิ่มขึ้นตามไป รวมทั้งการทำงานของต่อมน้ำเหลืองเช่นกัน เมื่อการทำงานของต่อมน้ำเหลืองเพิ่มขึ้นส่งผลให้น้ำเหลืองไหลเวียนทั่วร่างกายดีขึ้นระบบภูมิคุ้มกันจะมีความแข็งแรงเพิ่มมากขึ้นไปด้วย ในอีกกรณีหนึ่งเมื่อระบบไหลเวียนโลหิตดีขึ้นแล้วจะทำให้มั่นใจได้ว่าเซลล์ต่างๆ ในร่างกายจะได้รับออกซิเจนและสารอาหารที่เพียงพอ

ประโยชน์อื่นๆ ที่มาจากการนวดแบบสั่นสะเทือนคือ บรรเทาอาการปวดหลัง เสริมสร้างบุคลิกภาพ และลดเซลลูไลท์โดยการกระตุ้นการผลิตคอลลาเจน ซึ่งการใช้แรงสั่นสะเทือนในการนวดให้ผลเหมือนการออกกำลังกายในแบบต่างๆ ทั้งนี้ผลลัพธ์ที่ได้จึ้นอยู่กับกับรับประทานอาหารในแต่ละวันเช่นกัน ตัวอย่างเช่น มีความต้องการที่จะลดน้ำหนักโดยการใช้เครื่องนวดระบบสั่นเทือนเป็นเวลา 10 นาทีต่อวัน แต่ยังรับประทานอาหารที่มีแคลลอรี่สูงจึงไม่สามารถช่วยได้ ดังนั้นการรับประทานอาหารที่มีประโยชน์และปริมาณแคลลอรี่ต่ำ จะส่งผลต่อการลดน้ำหนักได้ดีกว่า

2.3.4) ช่วยในการลดน้ำหนัก อีกหนึ่งประโยชน์ที่ได้มาจากการนวดด้วยแรงสั่นสะเทือนคือสามารถเผาผลาญไขมันและลดน้ำหนักได้ โดยการนวดด้วยวิธีนี้จะกระตุ้นอัตราการเผาผลาญพลังงานในร่างกายให้เพิ่มขึ้นส่งผลให้อัตราการเผาผลาญแคลลอรี่เพิ่มขึ้นตามไป

2.3.5) กระตุ้นการสร้างฮอร์โมนที่มีประโยชน์ ผลของแรงสั่นสะเทือนที่ส่งไปยังกล้ามเนื้อส่วนต่างๆ ของร่างกายนั้นจะช่วยเสริมสร้างร่างกายให้มีการเพิ่มปริมาณการผลิตโกรทฮอร์โมน (Growth Hormone) ซึ่งฮอร์โมนนี้จะช่วยซ่อมแซมและเสริมสร้างกล้ามเนื้อและกระดูก อีกทั้งร่างกายจะลดปริมาณการผลิตคอร์ติซอล (ฮอร์โมนที่ก่อให้เกิดความเครียด) และผลิตเซโรโทนิน (ฮอร์โมนที่ก่อให้เกิดความสุข) ดังนั้นจึงเป็นเหตุผลหนึ่งที่ทำให้สามารถนอนหลับได้ดีขึ้นหลังจากได้รับการนวดแบบสั่นสะเทือน

2.3.6) ประโยชน์อื่นๆ การนวดด้วยระบบสั่นสะเทือนสามารถมีประโยชน์ในการรักษาบางอาการ เช่น อาการกล้ามเนื้ออ่อนแรง อาการปวดกล้ามเนื้อ โรคพาร์กินสัน อีกทั้งยังเป็นประโยชน์กับผู้สูงอายุที่ไม่สามารถออกกำลังกายแบบปกติได้

2.4) ผลเสียของการของการบำบัดด้วยการสั่นสะเทือน

การใช้แรงสั่นสะเทือนในการรักษาอาการปวดเมื่อยในบางกรณีอาจเป็นผลเสียกับร่างกายได้ หากระดับความรุนแรงของการสั่นมากเกินไป ซึ่งจะส่งผลให้เกิดการเจ็บปวดรุนแรงขึ้นในช่วงบริเวณรอบเอวและหลังตามมา ดังนั้นการบำบัดด้วยการใช้แรงสั่นสะเทือนควรปรึกษาแพทย์ เพื่อหาแนวทางที่เหมาะสมในการบำบัดอย่างถูกวิธีและปลอดภัยต่อร่างกาย โดยผู้ที่มีอาการดังต่อไปนี้ความหลีกเลี่ยงการใช้แรงสั่นสะเทือนในการรักษา

- ผู้ที่ได้รับยาป้องกันการจับตัวเป็นก้อนของเม็ดเลือด
- ผู้ที่เป็นโรคเบาหวานขั้นรุนแรง
- ผู้ที่มีอาการของโรคหัวใจ
- ผู้ที่ตั้งครรภ์

2.5) Microcontroller

ไมโครคอนโทรลเลอร์ถูกนำไปใช้ในระบบสมองกลฝังตัว คือ ระบบคอมพิวเตอร์ขนาดเล็กที่ถูกซ่อนอยู่ภายในเครื่องจักรกล เครื่องใช้ไฟฟ้า กลไกขนาดเล็กที่ควบคุมด้วยอิเล็กทรอนิกส์ เพื่อเพิ่มความชาญฉลาด หรือ เพิ่มประสิทธิภาพในการทำงานของเครื่องมือ ระบบสมองกลฝังตัวประกอบด้วย HARDWARE และ SOFTWARE ที่ทำงานร่วมกัน มีชื่อเรียกอย่างอื่นอีก เช่น smart devices (อุปกรณ์นำสมัย), intelligent (ระบบชาญฉลาด, ระบบปัญญาประดิษฐ์) หรือ automated equipment (ผลิตภัณฑ์อัตโนมัติ)

รูปที่ 2.1 Microcontroller ในปัจจุบัน

ที่มา : https://goo.gl/HSKPpJ

2.5.1) โครงสร้างโดยทั่วไป ของไมโครคอนโทรลเลอร์

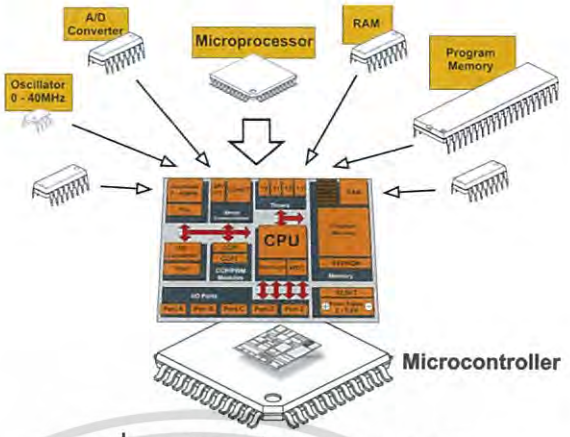
- หน่วยประมวลผลกลางหรือซีพียู (CPU : Central Processing Unit) ทำหน้าที่เป็นศูนย์กลางควบคุมการทำงานของระบบคอมพิวเตอร์ทั้งหมด โดยนำข้อมูลจากอุปกรณ์รับข้อมูลมาทำงาน ประมวลผลข้อมูลตามคำสั่งของโปรแกรม และส่งผลลัพธ์ออกไปหน่วยแสดงผล

- หน่วยความจำ (Memory) แบ่งเป็น 2 ส่วน คือ หน่วยความจำที่มีไว้สำหรับเก็บโปรแกรมหลัก (Program Memory) เปรียบเสมือนฮาร์ดดิสก์ของคอมพิวเตอร์ตั้งโต๊ะ คือข้อมูลใดๆ ที่ถูกเก็บไว้จะไม่สูญหายไปแม้ไม่มีไฟเลี้ยง อีกส่วนหนึ่งคือหน่วยความจำข้อมูล (Data Memory) ใช้เป็นเหมือนกระดาษทดในการคำนวณของซีพียู และเป็นที่เก็บข้อมูลชั่วคราวขณะทำงาน แต่หากไม่มีไฟเลี้ยงข้อมูลก็จะหายไปคล้ายกับหน่วยความแรม (RAM) ในเครื่องคอมพิวเตอร์ทั่วๆไป แต่สำหรับไมโครคอนโทรลเลอร์สมัยใหม่หน่วยความจำข้อมูลจะมีทั้งที่เป็นหน่วยความจำแรม ซึ่งข้อมูลจะหายไปเมื่อไม่มีไฟเลี้ยง และเป็นอีอีพรอม (EEPROM : Erasable Electrically Read-Only Memory) ซึ่งสามารถเก็บข้อมูลได้แม้ไม่มีไฟเลี้ยง

- ส่วนติดต่อกับอุปกรณ์ภายนอก หรือพอร์ต (Port) มี 2 ลักษณะคือ พอร์ตอินพุต (Input Port) และพอร์ตส่งสัญญาณหรือพอร์ตเอาต์พุต (Output Port) ส่วนนี้จะใช้ในการเชื่อมต่อกับอุปกรณ์ภายนอก ถือว่าเป็นส่วนที่สำคัญมาก ใช้ร่วมกันระหว่างพอร์ตอินพุต เพื่อรับสัญญาณ อาจจะด้วยการกดสวิตช์ เพื่อนำไปประมวลผลและส่งไปพอร์ตเอาต์พุต เพื่อแสดงผลเช่น การติดสว่างของหลอดไฟ

- ช่องทางเดินของสัญญาณ หรือบัส (BUS) คือเส้นทางการแลกเปลี่ยนสัญญาณข้อมูลระหว่างซีพียู หน่วยความจำและพอร์ต เป็นลักษณะของสายสัญญาณ จำนวนมากอยู่ภายในตัวไมโครคอนโทรลเลอร์ โดยแบ่งเป็นบัสข้อมูล (Data Bus) , บัสแอดเดรส (Address Bus) และบัสควบคุม (Control Bus)

- วงจรกำเนิดสัญญาณนาฬิกา นับเป็นส่วนประกอบที่สำคัญมากอีกส่วนหนึ่ง เนื่องจากการทำงานที่เกิดขึ้นในตัวไมโครคอนโทรลเลอร์ จะขึ้นอยู่กับการกำหนดจังหวะ หากสัญญาณนาฬิกามีความถี่สูง จังหวะการทำงานก็จะสามารถทำได้ถี่ขึ้นส่งผลให้ไมโครคอนโทรลเลอร์จัวนั้น มีความเร็วในการประมวลผลสูงตามไปด้วย

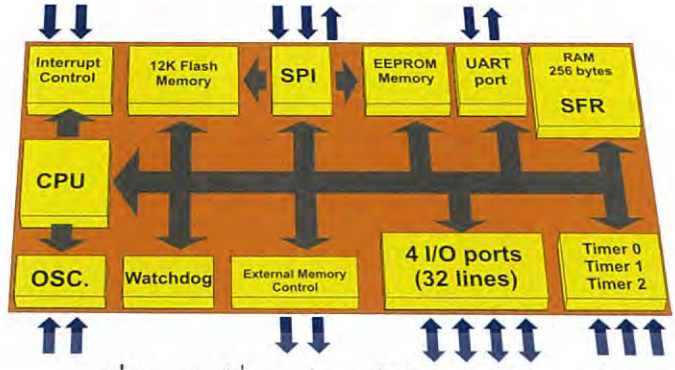


รูปที่ 2.2 โครงสร้าง Microcontroller

ที่มา : http://jumpstartinnovation.blogspot.com/2013/07/blog-post.html

นอกจากนี้ยังมีส่วนพิเศษอื่นๆ จะขึ้นอยู่กับกระบวนการผลิตของแต่ละบริษัทที่จะผลิตขึ้นมาใส่คุณสมบัติพิเศษลงไปเช่น

- **ADC** (Analog to Digital) ส่วนภาครับสัญญาณอนาล็อกแปลงไปเป็นสัญญาณดิจิตอล
- **DAC** (Digital to Analog) ส่วนภาคส่งสัญญาณดิจิตอลแปลงไปเป็นสัญญาณอนาล็อก
- **I2C** (Inter Integrate Circuit Bus) เป็นการสื่อสารอนุกรม แบบซิงโครนัส (Synchronous) เพื่อใช้ ติดต่อสื่อสาร ระหว่าง ไมโครคอนโทรลเลอร์ (MCU) กับอุปกรณ์ภายนอก ซึ่งถูกพัฒนาขึ้นโดยบริษัท Philips Semiconductors โดยใช้สายสัญญาณเพียง 2 เส้นเท่านั้น คือ serial data (SDA) และสาย serial clock (SCL) ซึ่งสามารถ เชื่อมต่ออุปกรณ์ จำนวนหลายๆ ตัว เข้าด้วยกันได้ ทำให้ MCU ใช้พอร์ตเพียง 2 พอร์ตเท่านั้น
- **SPI** (Serial Peripheral Interface) เป็นการเชื่อมต่อกับอุปกรณ์เพื่อรับส่งข้อมูลแบบซิงโครนัส (Synchronize) มีสัญญาณนาฬิกาเข้ามาเกี่ยวข้องระหว่างไมโครคอนโทรลเลอร์ (Microcontroller) หรือจะเป็นอุปกรณ์ภายนอกที่มีการรับส่งข้อมูลแบบ SPI อุปกรณ์ที่ทำหน้าที่เป็นมาสเตอร์ (Master) โดยปกติแล้วจะเป็ไมโครคอนโทรลเลอร์ หรืออาจกล่าวได้ว่าอุปกรณ์ Master จะต้องควบคุมอุปกรณ์ Slave ได้ โดยปกติตัว Slave มักจะเป็นไอซี (IC) หน้าที่พิเศษต่างๆ เช่น ไอซีอุณหภูมิ, ไอซีฐานเวลานาฬิกาจริง (Real-Time Clock) หรืออาจเป็นไมโครคอนโทรลเลอร์ที่ทำหน้าที่ในโหมด Slave ก็ได้
- **PWM** (Pulse Width Modulation) การสร้างสัญญาณพัลส์แบบสแควร์เวฟ ที่สามารถปรับเปลี่ยนความถี่และ Duty Cycle ได้เพื่อนำไปควบอุปกรณ์ต่างๆแช่น มอเตอร์
- **UART** (Universal Asynchronous Receiver Transmitter) ทำหน้าที่รับส่งข้อมูลแบบอะซิงโครนัสสำหรับมาตรฐานการรับส่งข้อมูลแบบ RS-232



รูปที่ 2.3 ฟังก์ชั่นภายในของไมโครคอนโทรลเลอร์

ที่มา : http://jumpstartinnovation.blogspot.com/2013/07/blog-post.html

2.5.2) ภาษาที่ใช้กับไมโครคอนโทรลเลอร์

- ภาษาเครื่อง (Machine Language) เป็นภาษาที่อยู่ในรูปแบบของรหัสเลขฐานสอง ไมโครคอนโทรลเลอร์สามารถเข้าใจภาษานี้ได้ทันที โดยไม่ต้องผ่านกระบวนการแปล แต่เป็นภาษาที่ยากต่อการเรียนรู้ เพราะอยู่ในรูปแบบของเลขฐานสอง และผู้ใช้ต้องมีความรู้เกี่ยวกับฮาร์ดแวร์เป็นอย่างดี แต่ข้อดีของภาษานี้ คือ มีขนาดเล็ก ทำงานได้รวดเร็ว และสามารถติดต่อกับฮาร์ดแวร์ได้โดยตรง

- ภาษา Assembly สร้างขึ้นมาเพื่อให้การเขียนโปรแกรมง่ายขึ้น ภาษา assembly ใช้คำในภาษาอังกฤษแทนรหัสเลขฐานสอง ในภาษาเครื่อง ดังนั้นในการใช้งาน จะต้องผ่านการแปลจากภาษา Assembly เป็นภาษาเครื่องก่อน ตัวแปลภาษา เรียกว่า Assembler โปรแกรมที่เขียนโดยภาพา assembly จะทำงานเร็วและมีขนาดเล็ก เพราะว่ามันสามารถเข้าถึง Hardware ได้โดยตรง เช่นเดียวกับภาษาเครื่อง แต่ทั้งนี้ขึ้นอยู่กับวิธีการเขียนของผู้เขียนด้วย

- Interpreter คือ ภาษาระดับสูงซึ่งใกล้เคียงกับภาษาของมนุษย์ โดยจะฝังตัวอยู่ในหน่วยความจำ และทำหน้าที่อ่านคำสั่งจากโปรแกรมขึ้นมาทีละคำสั่ง ทำการแปลเป็นภาษาเครื่องแล้วปฏิบัติตามคำสั่งนั้นๆ ตัวอย่างของ interpreter ที่รู้จักกันดีคือ ภาษา BASIC ข้อเสียของ interpreter คือ ทำงานได้ช้า เนื่องจากต้องแปลคำสั่งทีละคำสั่ง

- Compiler คือ ภาษาระดับสูงซึ่งทำหน้าที่แปลโปรแกรมที่เขียนขึ้นมาทั้งหมดให้เป็นภาษาเครื่อง จากนั้นจึงนำเอาโปรแกรมที่แปลเสร็จแล้วเข้าไปเก็บในหน่วยความจำ หลังจากนั้นจึงสั่งให้ไมโครคอนโทรลเลอร์ปฏิบัติตามคำสั่งนั้นๆ ทำให้การทำงานได้เร็วขึ้น ตัวอย่างเช่น ภาษา C เป็นต้น

2.5.3) ตระกูลต่างๆของไมโครคอนโทรลเลอร์

- ไมโคร คอนโทรลเลอร์ตระกูล PIC (บริษัทผู้ผลิต Microchip ไมโครชิป)
- ไมโคร คอนโทรลเลอร์ตระกูล MCS51 (บริษัทผู้ผลิต Atmel,Phillips)
- ไมโคร คอนโทรลเลอร์ตระกูล AVR (บริษัทผู้ผลิต Atmel)
- ไมโคร คอนโทรลเลอร์ตระกูล ARM7,ARM9 (บริษัทผู้ผลิต Atmel,Phillips,Analog Device,Sumsung,STMicroelectronics)
- ไมโคร คอนโทรลเลอร์ตระกูล Basic Stamp (บริษัทผู้ผลิต Parallax)
- ไมโคร คอนโทรลเลอร์ตระกูล PSOC (บริษัทผู้ผลิต CYPRESS)
- ไมโคร คอนโทรลเลอร์ตระกูล MSP (บริษัทผู้ผลิต Texas Intruments)
- ไมโคร คอนโทรลเลอร์ตระกูล 68HC (บริษัทผู้ผลิต MOTOROLA)
- ไมโคร คอนโทรลเลอร์ตระกูล H8 (บริษัทผู้ผลิต Renesas)
- ไมโคร คอนโทรลเลอร์ตระกูล RABBIT (บริษัทผู้ผลิต RABBIT SEMICONDUCTOR)
- ไมโคร คอนโทรลเลอร์ตระกูล Z80 (บริษัทผู้ผลิต Zilog)

2.6) มอเตอร์ไฟฟ้ากระแสตรง (DC Motor)

มอเตอร์ไฟฟ้าเป็นอุปกรณ์ที่นิยมใช้กันอย่างแพร่หลายในโรงงานต่างเป็นอุปกรณ์ที่ใช้ ควบคุมเครื่องจักรกลต่างๆในงานอุตสาหกรรมมอเตอร์มีหลายแบบหลายชนิดที่ใช้ให้เหมาะสมกับงาน จึงต้องทราบถึงความหมายและชนิดของมอเตอร์ไฟฟ้าตลอดคุณสมบัติการใช้งานของ มอเตอร์แต่ละชนิดเพื่อให้เกิดประสิทธิภาพสูงสุดในการใช้งานของมอเตอร์

มอเตอร์ไฟฟ้า (Motor) หมายถึงเป็นเครื่องกลไฟฟ้าชนิดหนึ่งที่เปลี่ยนแปลงพลังงานไฟฟ้า มาเป็นพลังงานกลมอเตอร์ไฟฟ้าที่ใช้พลังงานไฟฟ้าเปลี่ยนเป็นพลังงานกลมีทั้งพลังงานไฟฟ้า กระแสสลับและพลังงานไฟฟ้ากระแสตรง แบ่งออกเป็น 2 ชนิด

2.6.1) มอเตอร์ไฟฟ้ากระแสสลับ (Alternating Current Motor)

แบ่งออกเป็น 3 ชนิด

1. มอเตอร์ไฟฟ้ากระแสสลับชนิด 1 เฟส
   - สปลิทเฟสมอเตอร์ (Split-Phase motor)
   - คาปาซิเตอร์มอเตอร์ (Capacitor motor)
   - รีพลัชนั่ มอเตอร์ (Repulsion-type motor)
   - ยูนิเวอร์แซลมอเตอร์ (Universal motor)
   - เช็ดเดดโพลมอเตอร์ (Shaded-pole motor)

2. มอเตอร์ไฟฟ้ากระแสสลับชนิด 2 เฟส

3. มอเตอร์ไฟฟ้ากระแสสลับชนิด 3 เฟส

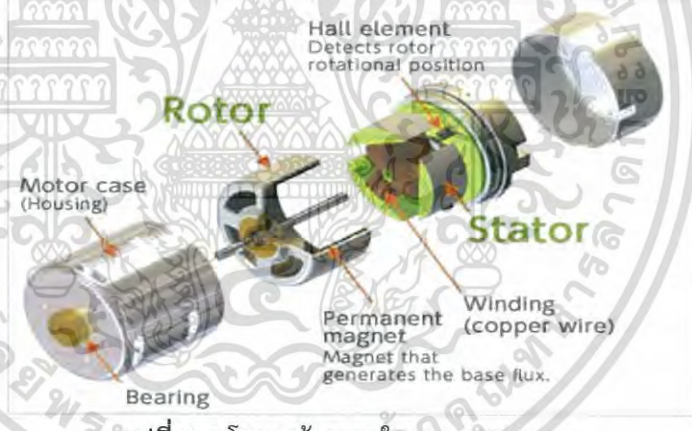
2.6.2) มอเตอร์ไฟฟ้ากระแสตรง (Direct Current Motor)

แบ่งออกเป็น 3 ชนิด

1. มอเตอร์แบบอนุกรมหรือเรียกว่าซีรีส์มอเตอร์ (Series Motor)
2. มอเตอร์แบบอนุขนานหรือเรียกว่าชันท์มอเตอร์ (Shunt Motor)
3. มอเตอร์ไฟฟ้าแบบผสมหรือเรียกว่าคอมปาวด์มอเตอร์ (Compound Motor)

มอเตอร์ไฟฟ้ากระแสตรงเป็นต้นกำลังขับเคลื่อนที่สำคัญอย่างหนึ่งในโรงงานอุตสาหกรรมเพราะมีคุณสมบัติที่ดีเด่นในด้านการปรับความเร็วได้ตั้งแต่ความเร็วต่ำสุดจนถึงสูงสุด นิยมใช้กันมากในโรงงานอุตสาหกรรม เช่นโรงงานทอผ้าโรงงานเส้นใยโพลีเอสเตอร์โรงงานถลุง โลหะหรือให้เป็นต้นกำลังในการขับเคลื่อนรถไฟฟ้าในการศึกษาเกี่ยวกับมอเตอร์ไฟฟ้า กระแสตรงจึงควรรู้จักอุปกรณ์ ต่างๆของมอเตอร์ไฟฟ้ากระแสตรงและเข้าใจหลักการทำงานของมอเตอร์ไฟฟ้ากระแสตรงแบบต่างๆ

ส่วนประกอบของมอเตอร์ไฟฟ้ากระแสตรงมีส่วนประกอบที่สำคัญ 2 ส่วน



รูปที่ 2.4 โครงสร้างภายในของ DC Motor
ที่มา : https://goo.gl/zw5rHL

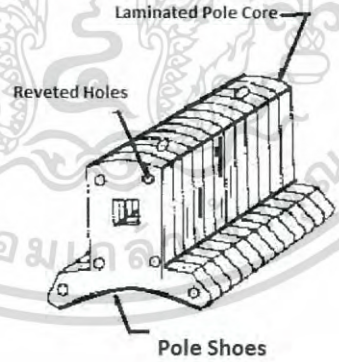
1. ส่วนที่อยู่กับที่หรือที่เรียกว่าสเตเตอร์ (Stator) ประกอบด้วย

- เฟรมหรือโยค (Frame or Yoke) เป็นโครงภายนอกทำหน้าที่เป็นทางเดินของเส้นแรงแม่เหล็กจากขั้วเหนือไปขั้วใต้ให้ครบวงจรและยึดส่วนประกอบอื่นๆให้แข็งแรงทำด้วย เหล็กหล่อหรือเหล็กแผ่นหนาม้วนเป็นรูปทรงกระบอก

รูปที่ 2.5 ส่วนที่อยู่กับที่หรือสเตเตอร์

ที่มา : http://www.jystatorrotor.com/cnblw/201631816542314976.jpg

- **ขั้วแม่เหล็ก (Pole)** ประกอบด้วย 2 ส่วนคือแกนขั้วแม่เหล็กและขดลวด ส่วนแรกแกนขั้ว (Pole Core) ทำด้วยแผ่นเหล็กบางๆกั้นด้วยฉนวนประกอบกัน เป็นแท่งยึดติดกับเฟรมส่วนปลายที่ทำเป็นรูปโค้งนั้นเพื่อโค้งรับรูปกลมของตัวโรเตอร์เรียกว่า ขั้วแม่เหล็ก (Pole Shoes) มีวัตถุประสงค์ให้ขั้วแม่เหล็กและโรเตอร์ใกล้ชิดกันมากที่สุดเพื่อให้เกิด ช่องอากาศน้อยที่สุดเพื่อให้เกิดช่องอากาศน้อยที่สุดจะมีผลให้เส้นแรงแม่เหล็กจากขั้วแม่เหล็กจาก ขั้วแม่เหล็กผ่านไปยังโรเตอร์มากที่สุดแล้วทำให้เกิดแรงบิดหรือกำลังบิดของโรเตอร์มากทำให้ มอเตอร์มีกำลังหมุน



รูปที่ 2.6 แกนขั้ว

ที่มา : http://www.polytechnichub.com/wp-content/uploads/2014/07/pole-core.gif

ส่วนที่สองขดลวดสนามแม่เหล็ก (Field Coil) จะพันอยู่รอบ ๆ แกนขั้วแม่เหล็กขดลวดนี้ทำหน้าที่รับกระแสจากภายนอกเพื่อสร้างเส้นแรงแม่เหล็กให้เกิดขึ้นและเส้นแรงแม่เหล็กนี้จะเกิดการหักล้างและเสริมกันกับสนามแม่เหล็กของอาเมเจอร์ทำให้เกิดแรงบิดขึ้น

รูปที่ 2.7 ขดลวดสนามแม่เหล็ก

ที่มา : https://4.imimg.com/data4/VX/RV/MY-24324888/motor-field-250x250.jpg

2. ตัวหมุน (Rotor) ตวัหมุนหรือเรียกว่าโรเตอร์ตัวหมุนนี้ทำให้เกิดกำลังงานมีแกนวางอยู่ในตลับลูกปืน (Ball Bearing) ซึ่งประกอบอยู่ในแผ่นปิดหัวท้าย (End Plate) ของมอเตอร์

ตัวโรเตอร์ประกอบด้วย 4 ส่วน

1. แกนเพลา (Shaft)
2. แกนเหล็กอาร์มาเจอร์ (Armature Core)
3. คอมมิวเตอร์ (Commutator)
4. ขอลวดอาร์มาเจอร์ (Armature Winding)

แกนเพลา (Shaft) เป็นตัวสำหรับยึดคอมมิวเตเตอร์และยึดแกนเหล็กอาร์มาเจอร์(Armature Core) ประกอบเป็นตัวโรเตอร์แกนเพลานี้จะวางอยู่บนแบริ่งเพื่อบังคับให้หมุนอยู่ในแนวนิ่งไม่มีการสั่นสะเทือนได้

แกนเหล็กอาร์มาเจอร์ (Armature Core) ทำด้วยแผ่นเหล็กบางอาบฉนวน (Laminated Sheet Steel) เป็นที่สำหรับพันขดลวดอาร์มาเจอร์ซึ่งสร้างแรงบิด (Torque)

คอมมิวเตเตอร์ (Commutator) ทำด้วยทองแดงออกแบบเป็นซี่แต่ละซี่มีฉนวน ไมก้า (mica) คั่นระหว่างซี่ของคอมมิวเตเตอร์ส่วนหัวซี่ของคอมมิวเตเตอร์จะมีร่องสำหรับใส่ปลายสายของขดลวดอาร์มาเจอร์ตัวคอมมิวเตเตอร์นี้อัดแน่นติดกับแกนเพลาเป็นรูปกลมทรงกระบอก มีหน้าที่สัมผัสกับแปรงถ่าน (Carbon Brushes) เพื่อรับกระแสจากสายป้อนเข้าไปยังขดลวด อาร์ มาเจอร์ เพื่อสร้างเส้นแรงแม่เหล็กอีกส่วนหนึ่งให้เกิดการหักล้างและเสริมกันกับเส้นแรงแม่เหล็กอีกส่วน ซ่งเกิดจากขดลวดขั้วแม่เหล็กดังกล่าวมาแล้วเรียกว่าปฏิกิริยามอเตอร์ (Motor action)

ขดลวดอาร์มาเจอร์ (Armature Winding) เป็นขดลวดพันอยู่ในร่องสลอท (Slot) ของแกนอาร์มาเจอร์ขนาดของลวดจะเล็กหรือใหญ่และจา นวนรอบจะมากหรือน้อยนั้นขึ้นอยู่กับการออกแบบของตัวโรเตอร์ชนิดนั้นๆ เพื่อที่จะใช้ให้เหมาะสมกับงานต่างๆ

รูปที่ 2.8 โครงสร้างของโรเตอร์

ที่มา : https://upload.wikimedia.org/wikipedia/commons/0/0d/Motor_rotor.jpg

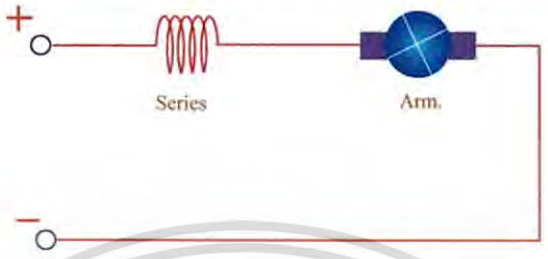
2.6.3) หลักการของมอเตอร์ไฟฟ้ากระแสตรง (Motor Action)

เมื่อเป็นแรงดันกระแสไฟฟ้าตรง เข้าไปในมอเตอร์ส่วนหนึ่งจะแปรงถ่านผ่านคอมมิวเตเตอร์เข้าไปในขดลวดอาร์มาเจอร์สร้างสนามแม่เหล็กขึ้นและกระแสไฟฟ้าอีกส่วนหนึ่งจะไหลเข้าไปในขดลวดสนามแม่เหล็ก (Field coil) สร้างขั้วเหนือ-ใต้ขึ้นจะเกิดสนามแม่เหล็ก 2 สนาม ในขณะเดียวกันตามคุณสมบัติของเส้นแรงแม่เหล็กจะไม่ตัดกันทิศทางตรงข้ามจะหักล้างกันและทิศทางเดียวจะเสริมแรงกันทำให้เกิดแรงบิดในตัวอาร์มาเจอร์ซึ่งวางแกนเพลาและแกนเพลานี้สวมอยู่กับตลับลูกปืนของมอเตอร์ทำให้อาร์มาเจอร์นี้หมุนได้ขณะที่ตวัอาร์มาเจอร์ทำหน้าที่หมุนได้นี้เรียกว่า โรเตอร์ (Rotor) ซึ่งหมายความว่าตัวหมุนการที่อำนาจเส้นแรงแม่เหล็กทั้งสองมีปฏิกิริยาต่อกันทำให้ขดลวดอาร์มาเจอร์หรื โรเตอร์หมุนไปนั้นเป็นไปตามกฎซ้ายของเฟลมมิ่ง (Fleming left hand rule)

2.6.4) ชนิดของมอเตอร์ไฟฟ้ากระแสตรง

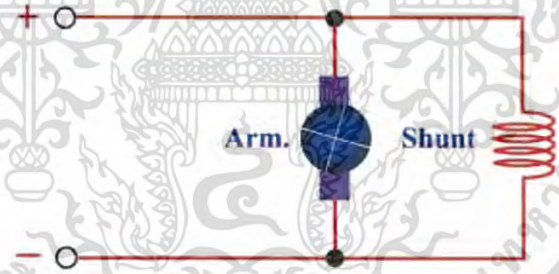
มอเตอร์แบบอนุกรม (Series Motor) คือมอเตอร์ที่ต่อขดลวดสนามแม่เหล็กอนุกรมกับอาร์เมเจอร์ของมอเตอร์ชนิดนี้ว่าซีรีสฟิลด์ (Series Field) มีคุณลักษณะที่ดีคือให้แรงบิดสูงนิยมใช้เป็นต้นกำลังของรถไฟฟ้ารถยกของเครนไฟฟ้าความเร็วรอบของมอเตอร์อนุกรมเมื่อไม่มีโหลดความเร็วจะสูงมาก แต่ถ้ามีโหลดมาต่อความเร็วก็จะลดลงตามโหลด โหลดมากหรือทำงานหนักความเร็วลดลง แต่ขดลวดของมอเตอร์ไม่เป็นอันตรายจากคุณสมบัตินี้จึงนิยมนำมาใช้กับเครื่องใช้ไฟฟ้าในบฺานหลายอย่าง เช่น เครื่องดูด

ฝุ่น เครื่องผสมอาหาร สว่านไฟฟ้าจักรเย็บผ้าเครื่องเป่าผม มอเตอร์กระแสตรงแบบอนุกรมใช้งานหนักได้ดี เมื่อใช้งานหนักกระแสจะมากความเร็วรอบจะลดลงเมื่อไม่มีโหลดมาต่อความเร็วจะสูงมากอาจเกิดอันตรายได้ดั้งนั้น เมื่อเริ่มสตาร์ทมอเตอร์แบบอนุกรมจึงต้องมีโหลดมาต่ออยู่เสมอ



รูปที่ 2.9 วงจรการทำงานของมอเตอร์ไฟฟ้ากระตรงแบบอนุกรม
ที่มา : http://202.129.59.73/tn/motor10-52/images/Series-Motor.jpg

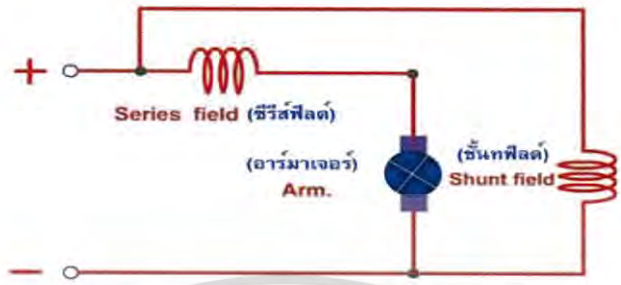
มอเตอร์ไฟฟ้ากระแสตรงแบบขนาน (Shunt Motor) หรือเรียกว่าชันท์มอเตอร์มอเตอร์แบบขนานนี้ขดลวดสนามแม่เหล็กจะต่อ (Field Coil) จะต่อขนานกับขดลวด ชุดอาเมเจอร์มอเตอร์แบบขนานนี้มีคุณลักษณะมีความเร็วคงที่แรงบิดเริ่มหมุนต่ำแต่ความเร็วรอบคงที่ชันท์มอเตอร์ส่วนมากเหมะกับงานดังนี้พัดลมเพราะพัดลมต้องการ ความเร็วคงที่และต้องการเปลี่ยนความเร็วได้ง่าย



รูปที่ 2.10 วงจรการทำงานมอเตอร์ไฟฟ้ากระแสตรงแบบขนาน
ที่มา : http://202.129.59.73/tn/motor10-52/images/Shunt-Motor.jpg

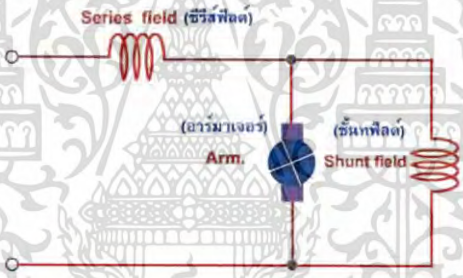
มอเตอร์ไฟฟ้ากระแสตรงแบบผสม (Compound Motor) หรือเรียกว่าคอมปาวด์มอเตอร์ มอเตอร์ไฟฟ้ากระแสตรงแบบผสมนี้จะนำคุณลักษณะที่ดีของมอเตอร์ไฟฟ้ากระแสตรง แบบขนาน และแบบอนุกรมมารวมกัน มอเตอร์แบบผสมมีคุณลักษณะพิเศษคือมีแรงบิดสูง (High Staring Torque) แต่ความเร็วรอบคงที่ตั้งแต่ยังไม่มีโหลดจนกระทั้งมีโหลดเต็มที่ มอเตอร์แบบผสมมีวิธีการต่อขดลวดขนานหรือขดลวดชันท์อยู่ 2 วิธี

วิธีที่ 1 ใช้ต่อขดลวดแบบชันท์ขนานกับอาเมเจอร์เรียกว่า ชอทชันท์ (Short Shunt Compound Motor) ดังรูปวงจร



รูปที่ 2.11 วงจรการทำงานมอเตอร์ไฟฟ้ากระตรงแบบชอร์ทชั้นท์คอมปาว์ดมอเตอร์
ที่มา : http://202.129.59.73/tn/motor10-52/images/Shunt-Motor.jpg

วิธีที่ 2 คือต่อขดลวด ขนานกับขดลวดอนุกรมและขดลวดอาเมเจอร์เรียกว่า ลองชั้นท์คอมปาวด์มอเตอร์ (Long shunt motor) ดังรูปวงจร

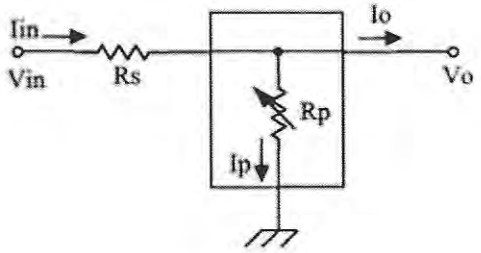


รูปที่ 2.12 วงจรการทำงานมอเตอร์ไฟฟ้ากระตรงแบบลองชั้นท์คอมปาว์ดมอเตอร์
ที่มา : http://202.129.59.73/tn/motor10-52/images/Shunt-Motor.jpg

2.7) ไอซีเร็กกูเลเตอร์ (IC Regulator)

2.7.1) เร็กกูเลเตอร์แบบขนาน (Shunt Regulator)

การทำงานของวงจรเร็กกูเลเตอร์แบบขนานดังรูปด้านล่างโดยมีแรงดันอินพุท VIN จ่าย ให้กับวงจร มีตัวต้านทาน RS ทำหน้าที่ในการจำกัดกระแสที่จะไหลผ่านวงจรทั้งหมด ตัวต้านทานที่ ปรับค่าได้ RP จะทำการปรับค่าเองโดยอัตโนมัติเพื่อให้แรงดันที่เอาท์พุทคงที่ตลอด สมการของ แรงดันเอาท์พุท VO = VIN – RS (IO + IP)

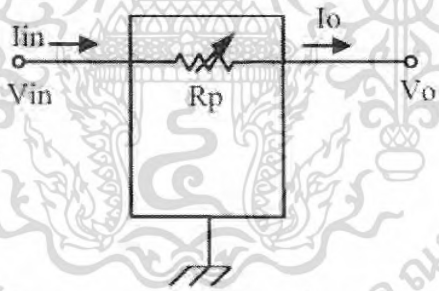


รูปที่ 2.13 แผนผังการทำงานของเร็กกูเลเตอร์แบบขนาน

ที่มา : http://www.academia.edu/8283067/%E0%B9 %81%E0%B8%A5_%E0%B8%9B_4

2.7.2) เร็กกูเลเตอร์แบบอนุกรม (Series Regulator)

หลักการทำงานของเร็กกูเลเตอร์แบบอนุกรมนี้แสดงในรูปด้านล่างโดยมีการจ่ายแรงดันที่ยังไม่ได้มีการเร็กกูเลทไปยัง RP โดย RP จะปรับค่าความต้านทานของตัวเองได้อัตโนมัติ ทำให้เกิด แรงดันตกคร่อมที่ RP ค่าหนึ่ง จะได้แรงดันเอาท์พุทเท่ากับแรงดันอินพุทลบด้วยแรงดันตกคร่อมใน ตัวเร็กกูเลเตอร์ ซึ่งผลของการปรับค่า RP ที่ ถูกต้องก็จะทำให้ได้แรงดันเอาท์พุทตามที่ต้องการ และ จากหลักการทำงานของเร็กกูเลเตอร์ชนิดนี้เองที่ได้นำมาประยุกต์ทำเป็นไอซีเร็กกูเลเตอร์เบอร์ ต่างๆ ทั้งเบอร์ 78XX เบอร์ 79XX และอื่นๆอีก



รูปที่ 2.14 แผนผังการทำงานของเร็กกูเลเตอร์แบบอนุกรม

ที่มา : http://www.academia.edu/8283067/%E0%B9%81%E0%B8%A5_%E0%B8%9B_4

2.7.3) แผนผังวงจรพื้นฐานของเร็กกูเลเตอร์แบบอนุกรม

แผนผังวงจรพื้นฐานของเร็กกูเลเตอร์ชนิดนี้สามารถแบ่งออกได้ 3 แบบ คือ

1. วงจรแรงดันอ้างอิง (Voltage Referent) ซึ่งเป็นส่วนที่เป็นอิสระต่อทั้งอุณหภูมิและ แรงดันที่จ่ายให้กับเร็กกูเลเตอร

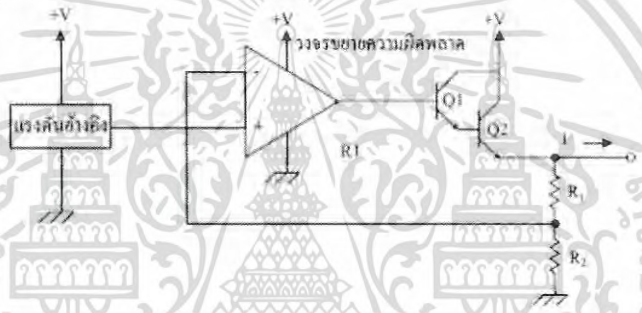
2. วงจรขยายความผิดพลาด (Error Amplifier) ทำหน้าที่คอยเปรียบเทียบแรงดันระหว่างแรงดันอ้างอิงและสัดส่วนของแรงดันเอาท์พุท ที่ป้อนกลับมาที่ขาอินเวอร์ติ้งของออปแอมป์

3. ซีรีส์พาสทรานซิสเตอร์ (Series Transistor) ทำหน้าที่จ่ายกระแสเอาท์พุทให้เพียงพอกับความต้องการของโหลด

เมื่อป้อนแรงดันอินพุทให้กับไอซีเร็กกูเลเตอร์แรงดันเอาท์พุทจะถูกป้อนมายังอินพุท โดย R1 และ R2 ทำหน้าที่เป็นวงจรแบ่งแรงดัน ซึ่งแรงดันที่ตกคร่อม R2 จะเป็นสัดส่วนกับแรงดันที่ เอาท์พุท วงจรขยายความผิดพลาดจะทำหน้าที่รักษาสัดส่วนของแรงดันอ้างอิงกับแรงดันที่ตกคร่อม R2 ให้เท่ากัน

ถ้าแรงดัน VR2 มากกว่า VREF วงจรขยายความผิดพลาดจะลดระดับการขยายสัญญาณเอาท์พุท ทำให้ทรานซิสเตอร์จ่ายกระแสลดลง เป็นผลให้แรงดันเอาท์พุทที่จ่ายให้โหลดลดลงด้วย

ถ้าแรงดัน VR2 น้อยกว่า VREF วงจร ขยายความผิดพลาดจะเพิ่มระดับการขยายสัญญาณเอาท์พุท ทำให้ทรานซิสเตอร์จ่ายกระแสเพิ่มขึ้น เป็นผลให้แรงดันเอาท์พุทที่จ่ายให้โหลดเพิ่มขึ้นด้วย

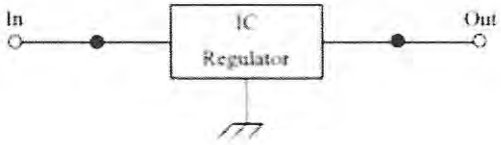


รูปที่ 2.15 แผนผังวงจรพื้นฐานของเร็กกูเลเตอร์แบบอนุกรม

ที่มา : http://www.academia.edu/8283067/%E0%B9%81%E0%B8%A5_%E0%B8%9B_4

2.7.4) ไอซีเร็กกูเลเตอร์สามขาชนิดจ่ายแรงดันคงที่

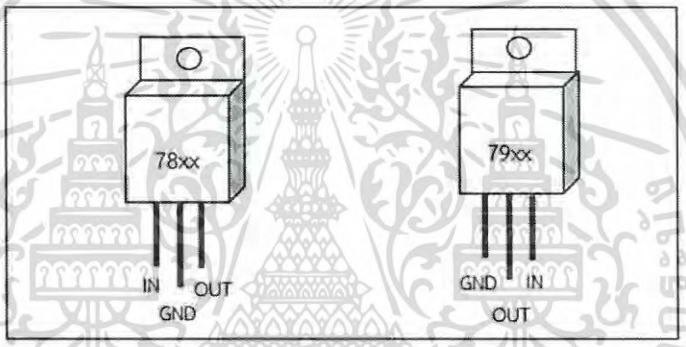
ไอซีเร็กกูเลเตอร์ภายในประกอบด้วยวงจรเร็กกูเลเตอร์แบบอนุกรม มีขาต่อใช้งาน 3 ขา ประกอบด้วยขา อินพุท เอาท์พุท และกราวด์ ซึ่งจะจ่ายแรงดันค่าใดค่าหนึ่งโดยเฉพาะ โดยรวมเอา ส่วนของวงจรป้อนกลับที่ประกอบด้วย R1 และ R2 เข้าไว้เป็นส่วนหนึ่งของไอซีซึ่งจุดนี้เองที่แตกต่างไปจากไอซีเร็กกูเลเตอร์ที่ปรับค่าได้



รูปที่ 2.16 การต่อไอซีเร็กกูเลตอร์ใช้งานแบบง่ายๆ

ที่มา : http://www.academia.edu/8283067/%E0%B9%81%E0%B8%A5_%E0%B8%9B_4

จุดเด่นของไอซีเร็กกูเลเตอร์ค่าคงที่นี้คือ สามารถต่อวงจรได้ง่ายไม่ต้องต่ออุปกรณ์ภายนอกเพิ่มเติมมากนักตัวอย่างวงจรการใช้งานในรูปข้างบน ในการต่อวงจรบางครั้งจำเป็นต้องต่อไอซี เร็กกูเลเตอร์ห่างจากแหล่งจ่ายไฟ อินพุทเกิน 5 เซนติเมตร จึงควรใส่ตัวเก็บประจุอิเล็กทรอไลต์ ขนาดประมาณ 10 ไมโครฟารัด 1 ตัวไว้ด้านอินพุท เพื่อป้องกันการเกิดออสซิลเลตที่ความถี่สูง ซึ่งจะทำให้วงจรขาดเสถียรภาพ เอาท์พุทที่ออกจากไอซีเร็กกูเลเตอร์จะได้แรงดันเอาท์พุทที่เรียบพอสมควรอยู่แล้ว แต่อาจจะใส่ตัวเก็บประจุที่มีค่าประมาณ 100 ไมโครฟารัด เพื่อช่วยปรับปรุง แรงดันให้เรียบขึ้น ถึงแม้ว่าแรงดันไอซีเร็กกูเลเตอร์ชนิดนี้จะให้แรงดัน เอาท์พุธคงที่มีเบอร์ให้เลือก แรงดันเอาท์พุทได้คงที่หลายเบอร์ เช่น 5 V, 5.2 V, 6V, 8V, 10V, 12V, 15V, 18V และ 24V กระแสเอาท์พุทตั้งแต่ 10 มิลลิแอมป์ – 3 แอมป์และมีให้เลือกทั้งชนิดเร็กกูเลเตอร์ไฟบวกและเร็กกูเลเตอร์ไฟลบ



รูปที่ 2.17 ตำแหน่งขาของIC Regulator เบอร์ 78xxและ79xx

ที่มา : http://www.chakchaielectronics.com/private_folder/regulators/2.jpg

2.8) ตัวต้านทาน (Resistor)

เป็นอุปกรณ์ที่ใช้ในการต้านทานการไหลของกระแสไฟฟ้า เพื่อทำให้กระแสและแรงดันภายในวงจร ได้ขนาดตามที่ต้องการ เนื่องจากอุปกรณ์ทางด้านอิเล็กทรอนิกส์แต่ละตัวถูกออกแบบให้ใช้แรงดันและกระแสที่แตกต่างกัน ดังนั้นตัวต้านทานจึงเป็นอุปกรณ์ที่มีบทบาทและใช้กันมากในงานด้านไฟฟ้าอิเล็กทรอนิกส์ เช่น วิทยุ, โทรทัศน์, คอมพิวเตอร์, เครื่องขยายเสียง ตลอดจนเครื่องมือเครื่องใช้ทางด้านไฟฟ้าอิเล็กทรอนิกส์ ฯลฯ เป็นต้น สัญลักษณ์ของตัวต้านทาน ที่ใช้ในการเขียนวงจรมีอยู่หลายแบบดังแสดงในรูป

รูปที่ 2.18 สัญลักษณ์ของตัวต้านทายชน

ที่มา : http://kpp.ac.th/elearning/elearning3/images-u/u-2/u201.jpg

2.8.1) ตัวต้านทานชนิดคาร์บอนผสม (Carbon Composition)

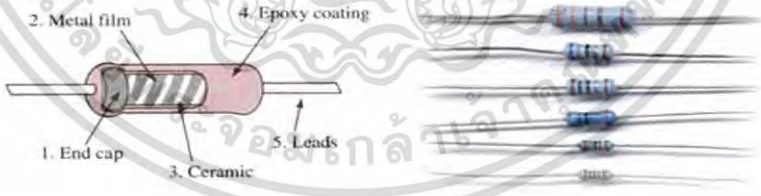
เป็นตัวต้านทานที่นิยมใช้กันมากซึ่งมีราคาถูก โครงสร้างตัวต้านทานชนิดนี้นั้นทำมาจากวัสดุที่มีคุณสมบัติเป็นตัวต้านทานผสมกันระหว่างผงคาร์บอนและผงของฉนวน อัตราส่วนผสมของวัสดุทั้งสองชนิดนี้จะทำให้ความต้านทานีค่าน้อยหรือมากเปลี่ยนแปลงได้ตามต้องการ บริเวรปลานทั้งสองข้างของตัวต้านทานต่อด้วยลวดตัวนำบริเวรด้านนอกของตัวต้านทานจะฉาบด้วยฉนวน

รูปที่ 2.19 ตัวต้านทานชนิดคาร์บอนผสม

ที่มา : http://kpp.ac.th/elearning/elearning3/images-u/u-2/u202.jpg

2.8.2) ตัวต้านทานแบบฟิล์มโลหะ (Metal Film)

ตัวต้านทานแบบฟิล์มโลหะทำมาจากแผ่นฟิล์มบางของแก้วและโลหะหลอมเข้าด้วยกันแล้วนำไปเคลือบที่เซรามิค ทำเป็นรูปทรงกระบอก แล้วตัดแผ่นฟิล์มที่เคลือบออกให้ได้ค่าความต้านทานตามที่ต้องการ ขั้นตอนสุดท้ายจะทำการเคลือบด้วยสารอีป๊อกซี (Epoxy) ตัวต้านทานชนิดนี้มีค่าความผิดพลาดบวกลบ 0.1 % ถึงประมาณ บวกลบ 2% ซึ่งถือว่ามีค่าความผิดพลาดน้อยมาก นอกจากนี้ยังทนต่อการเปลี่ยนแปลงอุณหภูมิจากภายนอกได้ดี สัญญาณรบกวนน้อยเมื่อเทียบกับตัวต้านทานชนิดอื่น ๆ



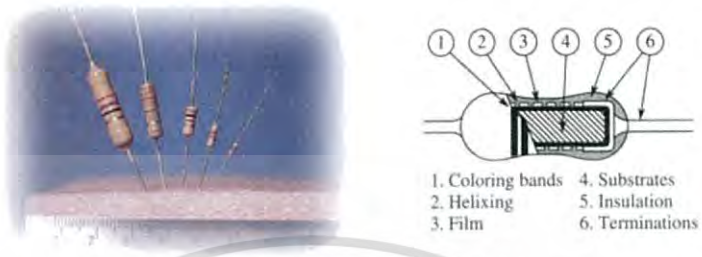
รูปที่ 2.20 ตัวต้านทานแบบฟิล์มโลหะ

ที่มา : http://kpp.ac.th/elearning/elearning3/images-u/u-2/u203.jpg

2.8.3) ตัวต้านทานแบบฟิล์มคาร์บอน (Carbon Film)

ตัวต้านทานแบบฟิล์มคาร์บอน เป็นตัวต้านทานแบบค่าคงที่โดยการฉาบผงคาร์บอน ลงบนแท่งเซรามิคซึ่งเป็นฉนวน หลังจากที่ทำการเคลือบแล้ว จะตัดฟิล์มเป็นวงแหวนเหมือนเกลียวน็อต ในกรณีที่เคลือบฟิล์มคาร์บอนในปริมาณน้อย จะทำให้ได้ค่าความต้านทานสูง แต่ถ้าเพิ่มฟิล์มคาร์บอนในปริมาณ

มากขึ้น จะทำให้ได้ค่าความต้านทานต่ำ ตัวต้านทานแบบฟิล์มโลหะมีค่าความผิดพลาด บวกลบ 5% ถึง บวกลบ 20% ทนกำลังวัตต์ตั้งแต่ 1/8 วัตต์ถึง 2 วัตต์ มีค่าความต้านทานตั้งแต่ 1 โอห์ม ถึง 100 เมกกะโอห์ม

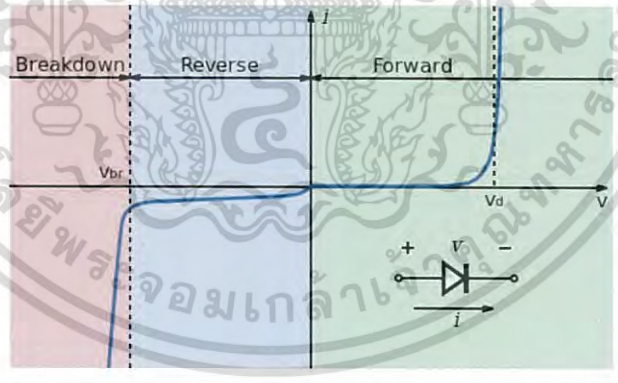


รูปที่ 2.21 ตัวต้านทานแบบฟิล์มคาร์บอน

ที่มา : http://kpp.ac.th/elearning/elearning3/images-u/u-2/u204.jpg

2.9) ไดโอด (Diode)

เป็นอุปกรณ์อิเล็กทรอนิกส์สารกึ่งตัวนำ มี 2 ขั่วคือ P และขั่ว N ถูกออกแบบมาเพื่อควบคุมทิศทางการไหลของประจุไฟฟ้ายอมให้กระแสไฟฟ้าไหลไปในทิศทางเดียวกัน และป้องกันกระแสการไหลกลับทิศทางเดิม หากมองหลักการทำงานก็เหมือนกับ วาล์วน้ำทิศทางเดียวไม่ยอมให้น้ำไหลกลับ ซึ่งนับเป็นประโยชน์อย่างมากในวงจรอิเล็กทรอนิกส์ เช่นวงจร บริดแปลงกระแสไฟฟ้ากระแสสลับ หรือป้องกันการสลับขั่วให้แก่วงจรอิเล็กทรอนิกส์



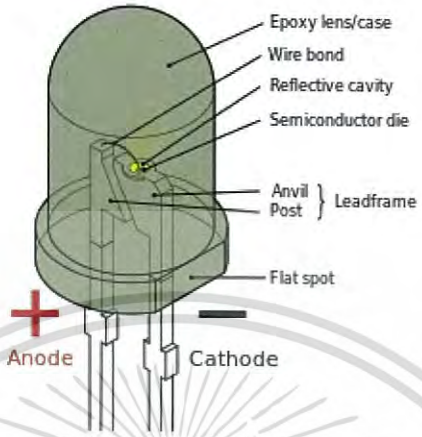
รูปที่ 2.22 แสดงความสัมพันธ์ระหว่าง V กับ I ของไดโอด

ที่มา : https://do.lnwfile.com/_/do/_raw/sn/ld/of.png

2.9.1) ประเภทของไดโอดชนิดต่างๆ

- ไดโอดเปล่งแสงหรือแอลอีดี (LED) เป็นไดโอนที่ใช้สารประเภทแกลเลี่ยมอาร์เซ็นไนต์ฟอสไฟต์ (Gallium Arsenide Phosphide) หรือสารแกลเลี้ยมฟอสไฟต์ มาเป็นสารกึ่งตัวนำ สารเหล่านี้มีคุณสมบัติ

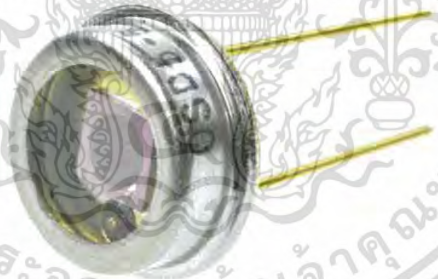
เมื่อได้รับไบอัสตรง จะเกิดแสงที่ตัวไดโอด LED ปัจจุบันนิยมนำมาใช้แทนหลอดไฟเพราะให้แสงสว่างที่มากกว่าและประหยัดไฟกว่า



รูปที่ 2.23 ไดโอดเปล่งแสง

ที่มา : https://en.wikipedia.org/wiki/Light-emitting_diode

- โฟโตไดโอด (Photo Diode) เป็นไดโอดที่อาศัยแสงจากภายนอก การต่อโฟโตไดโอดเพื่อใช้งานจะเป็นแบบไบอัสกลับ ทั้งนี้เพราะไม่ต้องการให้โฟโตไดโอดทำงานทันทีทันใด โฟโตไดโอดได้รับแสงสว่างจะเกิดกระแสรั่วไหลปริมาณกระแสรั่วไหลนี้เพิ่มขึ้นตามความเข้มของแสง



รูปที่ 2.24 โฟโตไดโอด

ที่มา : https://media.rs-online.com/t_large/F0846777-01.jpg

- **ไดโอดกำลัง (Power Diode)** นิยมนำมาใช้กับงานที่กำลังไฟฟ้าสูง กระแสสูงๆ ทนความต้อนได้อย่างดี นิยมนำมาใช้ประกอบเป็นวงจรเรียงกระแส ในอุปกรณ์อิเล็กทรอนิกส์ เป็นต้น

รูปที่ 2.25 ไดโอดกำลัง
ที่มา : https://elektronik-lavpris.dk/images/user_uploaded/2015/137845_1442252219.jpg

- **ซีเนอร์ไดโอด (Zener Diode)** เป็นอุปกรณ์สารกึ่งตัวนำวิธีการต่อจะต่อแบบไบอัสกลับ นิยมนำมาใช้ควบคุมแรงดันที่ต้องการแรงดันคงที่ เช่น อุปกรณ์แหล่งจ่ายไฟ หรือ Voltage Regulator

รูปที่ 2.26 ซีเนอร์ไดโอด
ที่มา : https://goo.gl/ZsBA3V

2.10) ทรานซิสเตอร์ (Transistor)

ทรานซิสเตอร์เป็นอุปกรณ์ที่พัฒนาจากไดโอด ซึ่งคุณสมบัติของทรานซิสเตอร หมายถึงสามารถนำไปใช้งานในด้านขยายสัญญาณให้มีขนาดใหญ่ขึ้น โดยการป้อนสัญญาณที่มีขนาดเล็กให้ทรานซิสเตอร์ ทรานซิสเตอร์ก็จะนำกระแสได้มากที่สามารถทำให้เกิดสัญญาณขนาดใหญ่ทางขาออกและทรานซิสเตอร์ยังเป็นอุปกรณ์สารกึ่งตัวนำที่สามารถทำหน้าที่ขยายสัญญาณไฟฟ้า เปิด/ปิด สัญญาณไฟฟ้า คงค่า

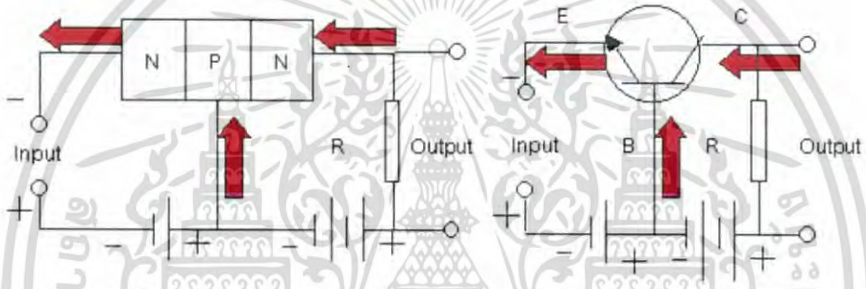
แรงดันไฟฟ้าหรือกล้ำสัญญาณไฟฟ้า (modulate) การทำงานของทราสซิสเตอร์เปรียบได้กับวาลว์ที่ถูกควบคุมด้วยสัญญาณไฟฟ้าขาเข้า เพื่อปรับขนาดกระแสไฟฟ้าขาออกที่มาจากแหล่งจ่ายแรงดัน

ทรานซิสเตอร์สามารถแบ่งตามโครงสร้างได้ 2 ประเภทคือ

1. ทรานซิสเตอร์ชนิด NPN
2. ทรานซิสเตอร์ชนิด PNP

2.10.1) การทำงานของทรานซิสเตอร์ชนิด NPN

การป้อนแรงดันไฟฟ้าให้กับทรานซิสเตอร์ชนิด NPN คือ การจ่ายไฟลบให้ขา E เมื่อเทียบกับที่จ่ายให้ขา B และจ่ายไฟบวกให้ขา C เมื่อเทียบกับไฟลบที่จ่ายให้ขา B มีทั้งไฟบวกและไฟลบ แต่การเทียบศักย์ Forward นั้นจะเทียบระหว่างขา B กับขา E ทำให้ขา B ซึ่งเป็นสาร P ได้รับแรงไฟ Forward คือเป็นไฟบวกเมื่อเทียบกับขา E เท่านั้นแสดงดังรูป

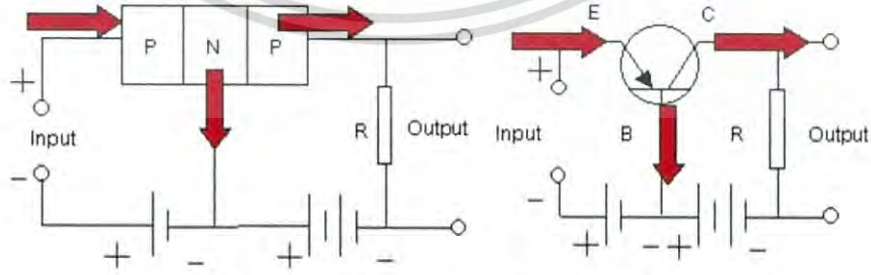


รูปที่ 2.27 การทำงานของทรานซิสเตอร์ชนิด NPN

ที่มา : http://phchitchai.wbvschool.net/wp-content/uploads/2011/02/image94.png

2.10.2) การทำงานของทรานซิสเตอร์ชนิด PNP

การป้อนแรงดันไฟฟ้าให้กับทรานซิสเตอร์ชนิด PNP โดยการจ่ายไฟบวกให้ขา E เมื่อเทียบ กับไฟลบที่จ่ายให้ขา B และจ่ายไฟลบเข้าขา C เมื่อเทียบกับไฟบวกที่จ่ายให้ขา B ทำให้ขา B มีทั้ง ไฟลบและไฟบวก ทำให้ขา B ซึ่งเป็นสาร N ได้รับ Forward Bias คือเป็นลบเมื่อเทียบกับขา E เท่านั้น แสดงดังรูป



รูปที่ 2.28 การทำงานของทรานซิสเตอร์ชนิด PNP

ที่มา : http://phchitchai.wbvschool.net/wp-content/uploads/2011/02/image95.png

2.11) ตัวเก็บประจุ (Capacitor)

เป็นอุปกรณ์อิเล็กทรอนิกส์อย่างหนึ่ง ทำหน้าที่เก็บพลังงานในสนามไฟฟ้า ที่สร้างขึ้นระหว่างคู่ฉนวน โดยมีค่าประจุไฟฟ้าเท่ากัน แต่มีชนิดของประจุตรงข้ามกัน บางครั้งเรียกตัวเก็บประจุนี้ว่าคอนเดนเซอร์(condenser)เป็นอุปกรณ์พื้นฐานสำคัญในงานอิเล็กทรอนิกส์ และพบได้แทบทุกวงจร

2.11.1) ชนิดอิเล็กโตรไลต์ (Electrolyte Capacitor)

เป็นที่นิยมใช้กันมากเพราะให้ค่าความจุสูง มีขั้วบวกลบ เวลาใช้งานต้องติดตั้งให้ถูกขั้ว โครงสร้างภายในคล้ายกับแบตเตอรี่ นิยมใช้กับงานความถี่ต่ำหรือใช้สำหรับไฟฟ้ากระแสตรง มีข้อเสียคือกระแสรั่วไหลและความผิดพลาดสูงมาก

รูปที่ 2.29 ตัวเก็บประจุชนิดอิเล็กโตรไลต์
ที่มา : https://goo.gl/PDZGqP

2.11.2) ชนิดเซรามิค (Ceramic Capacitor)

เป็นตัวเก็บประจุที่มีค่าไม่เกิน 1 ไมโครฟารัด นิยมใช้กันทั่วไปเพราะมีราคาถูก เหมาะสำหรับวงจรประเภทคัปปลิ้งความถี่วิทยุ ข้อเสียของตัวเก็บประจุชนิดเซรามิคคือมีการสูญเสียมาก

รูปที่ 2.30 ตัวเก็บประจุชนิดเซรามิค
ที่มา : https://goo.gl/LqakLX

2.12) อุปกรณ์ รับ-ส่ง สัญญาณบลูทูธ (Bluetooth module)

ระบบสื่อสารของอุปกรณ์อิเล็คโทรนิคแบบสองทาง ด้วยคลื่นวิทยุระยะสั้น (Short-Range Radio Links) โดยปราศจากการใช้สายเคเบิ้ล หรือ สายสัญญาณเชื่อมต่อ และไม่จำเป็นจะต้องใช้การเดินทางแบบเส้นตรงเหมือนกับอินฟราเรด ซึ่งถือว่าเพิ่มความสะดวกมากกว่าการเชื่อมต่อแบบอินฟราเรด ที่ใช้ในการเชื่อมต่อระหว่างโทรศัพท์มือถือกับอุปกรณ์ในโทรศัพท์เคลื่อนที่รุ่นก่อนๆ และในการวิจัย ไม่ได้มุ่งเฉพาะการส่งข้อมูลเพียงอย่างเดียว แต่ยังศึกษาถึงการส่งข้อมูลที่เป็นเสียง เพื่อใช้สำหรับ Headset บนโทรศัพท์มือถือด้วย

รูปที่ 2.31 อุปกรณ์ รับ-ส่ง สัญญาณบลูทูธ

ที่มา : http://www.neodigitalperu.com/wp-content/uploads/2018/02/Bluetooth-B5-3.jpg

2.12.1) การทำงานของ Bluetooth

Bluetooth จะใช้สัญญาณวิทยุความถี่สูง 2.4 GHz. แต่จะแยกย่อยออกไป ตามแต่ละประเทศ อย่างในแถบยุโรปและอเมริกา จะใช้ช่วง 2.400 ถึง 2.4835 GHz. แบ่งออกเป็น 79 ช่องสัญญาณ และจะใช้ช่องสัญญาณที่แบ่งนี้ เพื่อส่งข้อมูลสลับช่องไปมา 1,600 ครั้งต่อ 1 วินาที ส่วนที่ญี่ปุ่นจะใช้ความถี่ 2.402 ถึง 2.480 GHz. แบ่งออกเป็น 23 ช่อง ระยะทำการของ Bluetooth จะอยู่ที่ 5-10 เมตร โดยมีระบบป้องกันโดยใช้การป้อนรหัสก่อนการเชื่อมต่อ และ ป้องกันการดักสัญญาณระหว่างสื่อสาร โดยระบบจะสลับช่องสัญญาณไปมา จะมีความสามารถในการเลือกเปลี่ยนความถี่ที่ใช้ในการติดต่อเองอัตโนมัติ โดยที่ไม่จำเป็นต้องเรียงตามหมายเลขช่อง ทำให้การดักฟังหรือลักลอบขโมยข้อมูลทำได้ยากขึ้น

โดยหลักของบลูทูธจะถูกออกแบบมาเพื่อใช้กับอุปกรณ์ที่มีขนาดเล็ก เนื่องจากใช้การขนส่งข้อมูลในจำนวนที่ไม่มาก อย่างเช่น ไฟล์ภาพ, เสียง, แอพพลิเคชั่นต่างๆ และสามารถเคลื่อนย้ายได้ง่าย ขอให้อยู่

ในระยะที่กำหนดไว้เท่านั้น (ประมาณ 5-10 เมตร) นอกจากนี้ยังใช้พลังงานต่ำ กินไฟน้อย และสามารถใช้งานได้นาน โดยไม่ต้องนำไปชาร์จไฟบ่อยๆ ด้วย

ส่วนความสามารถการส่งถ่ายข้อมูลของ Bluetooth จะอยู่ที่ 1 Mbps (1 เมกกะบิตต่อวินาที) และคงจะไม่มีปัญหาอะไรมากกับขนาดของไฟล์ที่ใช้กันบนโทรศัพท์มือถือหรือการใช้งานแบบทั่วไป แต่ถ้าเป็นข้อมูลที่มีขนาดใหญ่คงจะช้าเกินไปและถ้าถูกนำไปเปรียบกับ Wireless LAN (WLAN) ความสามารถของ Bluetooth จะห่างชั้นกันมาก ซึ่งในส่วนของ WLAN ยังมีระยะการรับ-ส่งที่ไกลกว่า แต่ข้อได้เปรียบของ Bluetooth จะอยู่ที่ขนาดที่เล็กกว่า การติดตั้งทำได้ง่ายกว่าและที่สำคัญการใช้พลังงานก็น้อยกว่ามาก อยู่ที่ 0.1 วัตต์ หากเทียบกับคลื่นมือถือแล้ว ยังห่างกันอยู่หลายเท่าเหมือนกัน

2.13) สัญญาณ PWM (Pulse Width Modulation)

Pulse Width Modulation เป็นสัญญาณพัลส์ที่มีค่าความถี่คงที่แต่สามารถเปลี่ยนแปลงความกว้างของพัลส์ได้ ซึ่งลักษณะสัญญาณจะเป็นรูปสี่เหลี่ยม

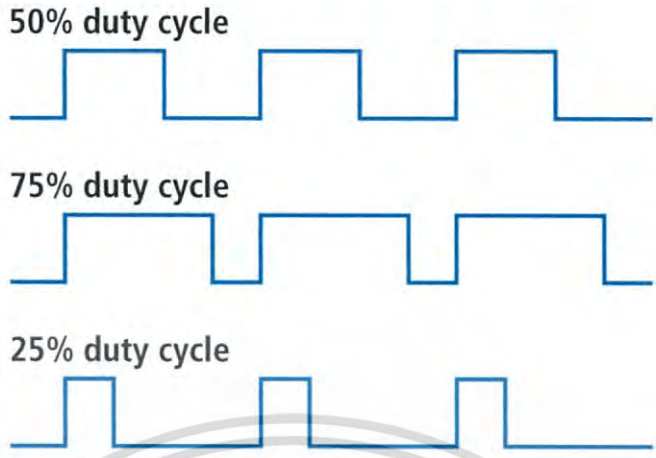
รูปที่ 2.32 สัญญาณ PWM

ที่มา : https://www.thitiblog.com/blog/6142

ในระบบ digital นั้นจะมีสัญญาณแค่ High และ Low ถ้าต้องการจะนำระบบ digital ไปควบคุมอุปกรณ์ที่ต้องควบคุมด้วยสัญญาน analog เช่น การควบคุมความเร็วมอเตอร์, การควบความความสว่างของหลอดไฟ ฯลฯ ซึ่งในระบบ digital จะไม่สามารถควบคุมอุปกรณ์พวกนี้ได้ตรงๆ เราจะต้องใช้ PWM ในการควบคุม โดยจะปรับคาบเวลาของ High-Low (Duty cycle)

2.13.1) Duty Cycle

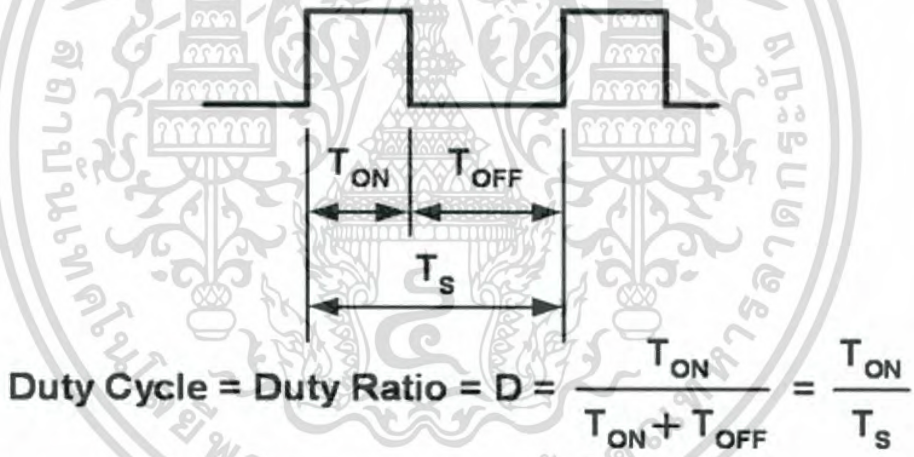
Duty Cycle คือ อัตราส่วนของสัญญาณดิจิตอลระหว่าง High และ Low โดยค่า Duty Cycle นั้นจะวัดเป็นเปอร์เซ็น (%)



รูปที่ 2.33 Duty Cycle

ที่มา : https://learn.sparkfun.com/tutorials/pulse-width-modulation/duty-cycle

การคำนวณหาอัตราส่วนของ Duty Cycle นั้นสามารถหาจากจากสมการรูปนี้



รูปที่ 2.34 แสดงการคำนวณหา Duty Cycle

ที่มา : http://lights.ofweek.com/2012-01/ART-220001-8300-28595660.html

2.14) ภาษาซี (C Language)

ภาษา C ถูกสร้างขึ้นครั้งแรกโดย Dennis M.Ritchie ซึ่งทำงานอยู่ Bell Telephone Laboratories, Inc. (ปัจจุบันนี้คือ AT&T Bell Laboratories) ประมาณปี ค.ศ.1970 โดย Ritchie พัฒนาภาษา C มาจากภาษา BCPL และภาษา B ซึ่งในระยะแรกนี้ภาษา C ถูกนำมาใช้ภายใน Bell Laboratories เท่านั้น จนกระทั่งปี ค.ศ.978 Brian W.Kerninghan และ Dennis M. Ritchie ได้กำหนดนิยาม ลักษณะ และรายละเอียดของภาษา C ขึ้น โดยเขียนหนังสือชื่อว่า "The C Programming Language" (สำนักพิมพ์ Prentice Hall) ออกมาเป็นเล่มแรกต่อมาบริษัทคอมพิวเตอร์ต่าง ๆ ได้เริ่มสนใจและค้นคว้าพัฒนาภาษา C โดยอ้างอิงภาษา C ของ Kernighan และ Ritchie ทำให้มีการพัฒนา C compiler และ C interpreter ขึ้นมาเพื่อให้สามารถใช้กับเครื่องคอมพิวเตอร์ได้หลายๆชนิดและสามารถใช้กับโปรแกรมต่างๆที่บริษัทผลิตขึ้นเป็นการค้า จนกระทั่งปี ค.ศ.1985 ภาษา C ก็ได้รับความนิยมแพร่หลายไปทั่วโลก ซึ่งในช่วงนั้นภาษา C ที่ใช้กันอยู่มีมากมายหลายชนิด แล้วแต่บริษัทต่างๆจะสร้างขึ้นซึ่งยังขาดมาตรฐานร่วมกัน ดังนั้นในปี ค.ศ.1988 Kernighan และ Ritchie จึงได้ร่วมกับสถาบัน ANSI (American National Standards Institute) ได้กำหนดนิยาม ลักษณะและกฎเกณฑ์ของภาษา C ที่เป็นมาตรฐานขึ้นเรียกว่า "ANSI C" ซึ่งปัจจุบันนี้บริษัทที่ผลิตภาษา C ไม่ว่าจะเป็นบริษัท Microsoft และบริษัท Borland ต่างก็ใช้มาตรฐานของ ANSI C เพื่อผลิตภาษา C รุ่นต่างๆต่อไป

รูปที่ 2.35 ภาษาซี

ที่มา : http://marcuscode.com/lang/c

2.14.1) คำสั่งสำคัญในภาษาซี

- #include ใช้สำหรับนำเข้าโค้ดจากไฟล์อื่นเข้ามาใช้งานในโปรแกรม ซึ่งประกอบไปด้วยฟังก์ชันต่างๆ ที่จำเป็นในการเขียนโปรแกรม เช่น ฟังก์ชันสำหรับแสดงผลออกทางหน้าจอ หรือรับค่าจากคีย์บอร์ด เป็นต้น หรือสามารถสร้างไฟล์ .h เองได้เพื่อนำมาใช้ในโปรแกรม

```
ตัวอย่างเช่น    #include <stdio.h>
```

- Comment เป็นส่วนของโค้ดที่ไม่มีผลต่อการทำงานของโปรแกรม มันถูกใช้เพื่ออธิบายโปรแกรมสำหรับเป็นภาษามนุษย์เข้าใจในภาษาซีซึ่งต้องมี “//” (Slash) สองตัวอยู่ข้างหน้าเสมอ

```
ตัวอย่างเช่น    // This is my comment
```

- เซมิโคลอน (Semicolon) (;) เป็นสัญลักษณ์ที่ใช้เพื่อแบ่งแยกคำสั่งภายในโปรแกรม ซึ่งมันหมายถึงการจบคำสั่งนั้นๆ เซมิโคลอนใช้ในภาษาต่างๆ และมันเป็นสิ่งที่บังคับ เพื่อให้ตัว คอมไพเลอร์ของภาษาสามารถแยกแยะคำสั่งในการทำงานได้ หากไม่มีจะส่งผลให้โปรแกรม ทำงานผิดพลาดได้

```
ตัวอย่างเช่น    int B;
                int Z = 1; A = 10;
                printf ("%d", a+b);
```

- Keyword เป็นกลุ่มคำที่ถูกสงวนไว้โดยเราไม่สามารถใช้คำเหล่านี้ในการประกาศเป็นชื่อตัวแปร ฟังก์ชัน ซึ่งในโปรแกรมทุกภาษาต่างก็มี keyword

ตัวอย่าง keyword ที่ใช้ในภาษาซี

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

- Arithmetic operators (+, -, *, /, %) ตัวดำเนินการทางคณิตศาสตร์ คือตัวดำเนินการที่ใช้เพื่อกระทำการดำเนินการทางคณิตศาสตร์ระหว่างตัวแปรหรือค่าคงที่ เช่น การบวก การลบ การคูณ และการหาร สำหรับในการเขียนโปรแกรมในภาษา C นั้นจะมีตัวดำเนินการสำหรับการหารเอาเศษ (Modulo) เพิ่มเข้ามา

ตัวอย่างตัวดำเนินการที่ใช้ในภาษาซี

Symbol	Name	Example
+	Addition	c = a + b
-	Subtraction	c = a - b
*	Multiplication	c = a * b
/	Division	c = a / b
%	Modulo	c = a % b

- Relational และ comparison operators (==, ! =, >, <, >=, <=) ตัวดำเนินการความสัมพันธ์และเปรียบเทียบ คือตัวดำเนินการที่ถูกใช้เพื่อประเมินค่า true และ false ระหว่างสองตัวถูกดำเนินการซึ่งขึ้นกับเงื่อนไขและความสัมพันธ์ของมัน

ตัวอย่างตัวดำเนินการที่ใช้ในภาษาซี

Operator	Example	Result
==	a == b	true if a equal to b, otherwise false
!=	a != b	true if a not equal to b, otherwise false
<	a < b	rue if a less than b, otherwise false
>	a > b	true if a greater than b, otherwise false
<=	a <= b	rue if a less than or equal to b, otherwise false
>=	a >= b	true if a greater than or equal to b, otherwise false

- If เป็นคำสั่งที่ใช้ในการตรวจสอบว่าเงื่อนไขเป็นจริงหรือไม่ โดยการใช้ตัวดำเนินการต่างๆ ในการสร้างเงื่อนไขหรือ expression ถ้าผลลัพธ์เป็นจริง โปรแกรมจะทำงานในบล็อคของคำสั่ง if ที่เราได้กำหนดขึ้น และถ้าหากผลลัพธ์ไม่เป็นจริงโปรแกรมจะข้ามการทำงานในบล็อคคำสั่งไป

ตัวอย่างเช่น

```
if (3 < 1)
{printf ("block 2 is executed.\n");}

int a = 10;
if (a % 2 == 0)
{printf ("block 3 is executed.\n");}
```

- If else เป็นคำสั่งมีการเพิ่มเงื่อนไข else if เข้ามาสำหรับการสร้างเงื่อนไขแบบหลายทางเลือก นั่นหมายถึงในบล็อคของคำสั่งจะทำงานเมื่อเงื่อนไขไม่ต้องกับเงื่อนไขก่อนหน้าแต่ตรงกับเงื่อนไขปัจจุบัน นอกจากนี้ยังมีคำสั่ง else ซึ่งเป็นบล็อคของคำสังที่จะทำงานถ้าหากไม่มีเงื่อนไขใดเป็นจริงเลย

ตัวอย่างเช่น

```
int score;
scanf ("%d", &score);
if (score < 50)
printf ("Your grade is F");
else if (score < 60)
printf ("Your grade is D");
else if (score < 70)
printf ("Your grade is C");
else if (score < 80)
printf ("Your grade is B");
else
printf ("Congratulation! You got grade A");
```

- Switch case เป็นคำสั่งเงื่อนไขที่ทำงานคล้ายกับ if แต่ส่วนมากจะใช้สำหรับการเปรียบเทียบกับค่าคงที่และต้องเป็นข้อมูลประเภท Integer หรือ Char เท่านั้น

ตัวอย่างเช่น

```
int number = 5;
switch (number) {
case 1:
printf ("n is 1");
break;
case 2:
printf ("n is 2");
break;
case 3:
printf ("n is 3");
break;
default:
printf ("Unknown"); }
```

- while loop คำสั่งของการทำงานซ้ำ ซึ่งสามารถให้เราประมวลผลส่วนของโปรแกรมที่ต้องการได้คำสั่งวนซ้ำที่ง่ายที่สุดคือคำสั่ง while loop โดยการทำงานของคำสั่ง while loop จะทำงานเมื่อเงื่อนไขเป็นจริงและจะสิ้นสุดการทำงานจนกว่าเงื่อนไขจะเป็นเท็จ

ตัวอย่างเช่น

```
#include <stdio.h>
int main () {
int n = 1;
while (n <= 10) {
printf ("%d, ", n);
n++;
}
return 0;}
```

-for loop เป็นคำสั่งวนซ้ำที่เราสามารถกำหนดค่าเริ่มต้นเงื่อนไขและการเปลี่ยนแปลงของค่าได้ในตอนแรกครั้งเดียว โดยทั่วไปแล้วมักจะใช้คำสั่ง for loop กับการวนซ้ำในจำนวนรวบที่แน่นอนหรือมันมักจะถูกใช้กับอาเรย์

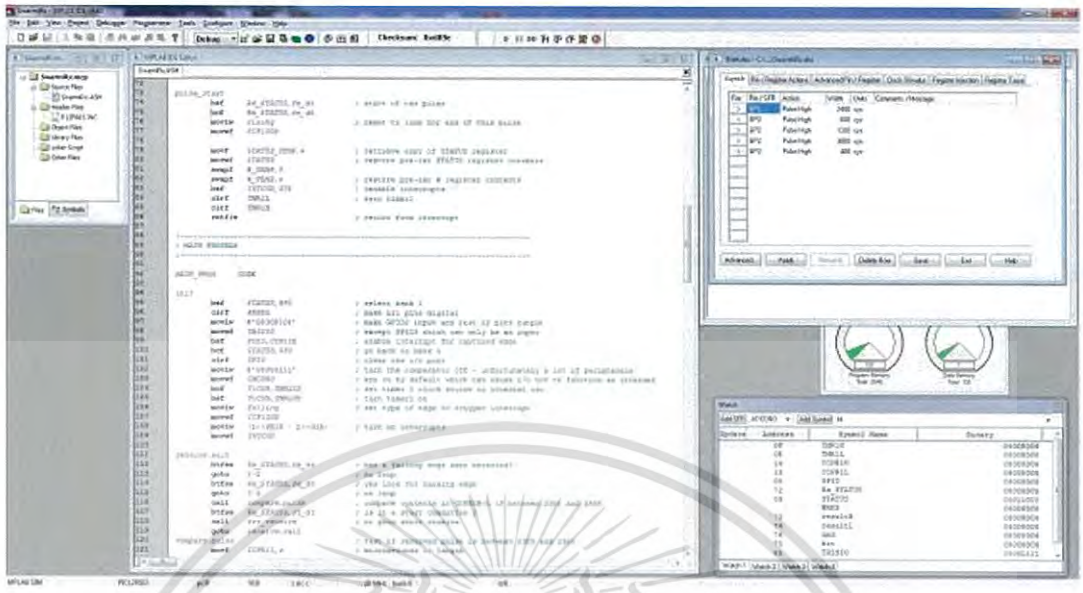
ตัวอย่างเช่น

```
int i;
for (i = 0; i <= 10; i++) {
printf ("%d, ", i);
}
```

2.15) โปรแกรม MPLAB IDE

รูปที่ 2.36 สัญลักษณ์โปรแกรม MPLAB IDE
ที่มา : https://goo.gl/ueYS4o

โปรแกรม MPLAB IDE เป็นเครื่องมือสำหรับพัฒนาโปรแกรมไมโครคอนโทรลเลอร์ซึ่งถูกพัฒนาโดยบริษัท Microchip ซึ่ง MPLAB IDE เป็น Open-Source Platform ซึ่งพัฒนาภายใต้ The NetBeans Platform ซึ่งเป็น Platform ชื่อดังในการพัฒนาโปรแกรม Java, C/C++ และอื่นๆเป็นเครื่องมือสำหรับพัฒนาภาษาซี ดีบักเกอร์ และอีดิเตอร์ไว้ในตัวเดียวกัน เพื่อให้สามารถเขียนโปรแกรมภาษา C ลงบนไมโครคอนโทรลเลอร์ได้



รูปที่ 2.37 ตัวอย่างหน้าต่างโปรแกรมที่กำลังใช้งาน

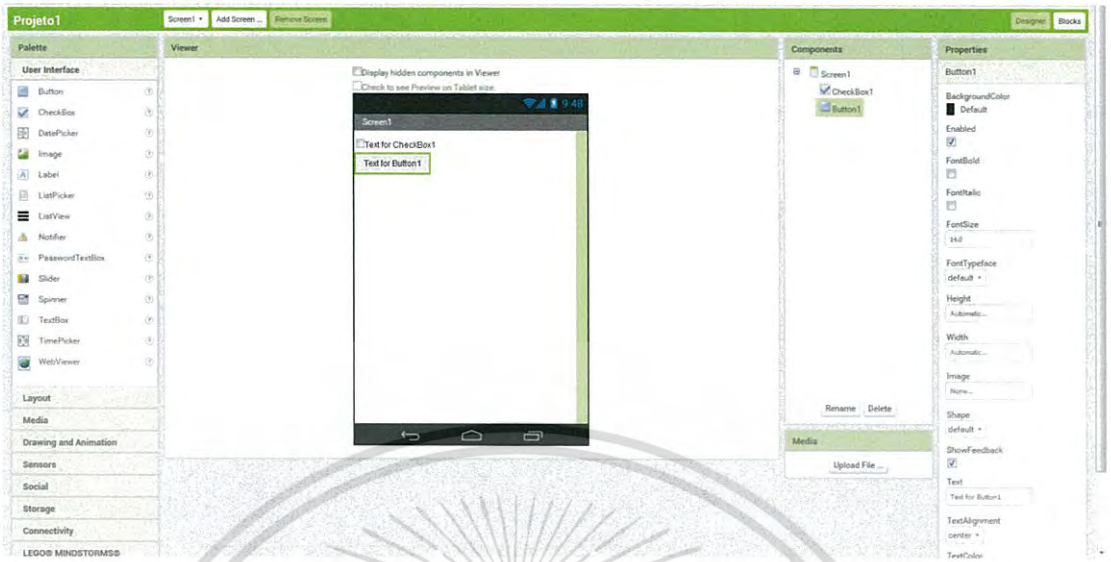
ที่มา : https://www.file-extensions.org/imgs/app-picture/11174/mplab-ide.jpg

2.16) โปรแกรม App Inventor2

โปรแกรม App Inventor2 เป็นเครื่องมือที่ใช้สำหรับพัฒนาแอพพลิเคชั่น สำหรับสมาร์ทโฟนและแท็บเล็ตที่เป็นระบบปฏิบัติการ Android ซึ่งบริษัท Google ร่วมมือกับ MIT พัฒนาโปรแกรม App inventor2 ขึ้น App inventor ใช้หลักการคล้ายๆ กับ Scratch แต่ซับซ้อนกว่า โดยลักษณะการเขียนโปรแกรมแบบ Visual Programming คือ เขียนโปรแกรมด้วยการต่อบล็อกคำสั่ง เน้นการออกแบบเพื่อแก้ปัญหา (problem solving) ด้วยการสร้างโปรแกรมที่ผู้เรียนสนใจ บนโทรศัพท์มือถือสมาร์ทโฟน App inventor จึงเป็นอีกโปรแกรมหนึ่ง ที่เหมาะสำหรับใช้ในการสอนเขียนโปรแกรม ให้นักเรียนในระดับมัธยมปลาย หรือระดับมหาวิทยาลัย โดยเฉพาะผู้ที่ไม่เคยเขียนโปรแกรมมาก่อนหรือไม่ได้เรียนอยู่ในสายคอมพิวเตอร์

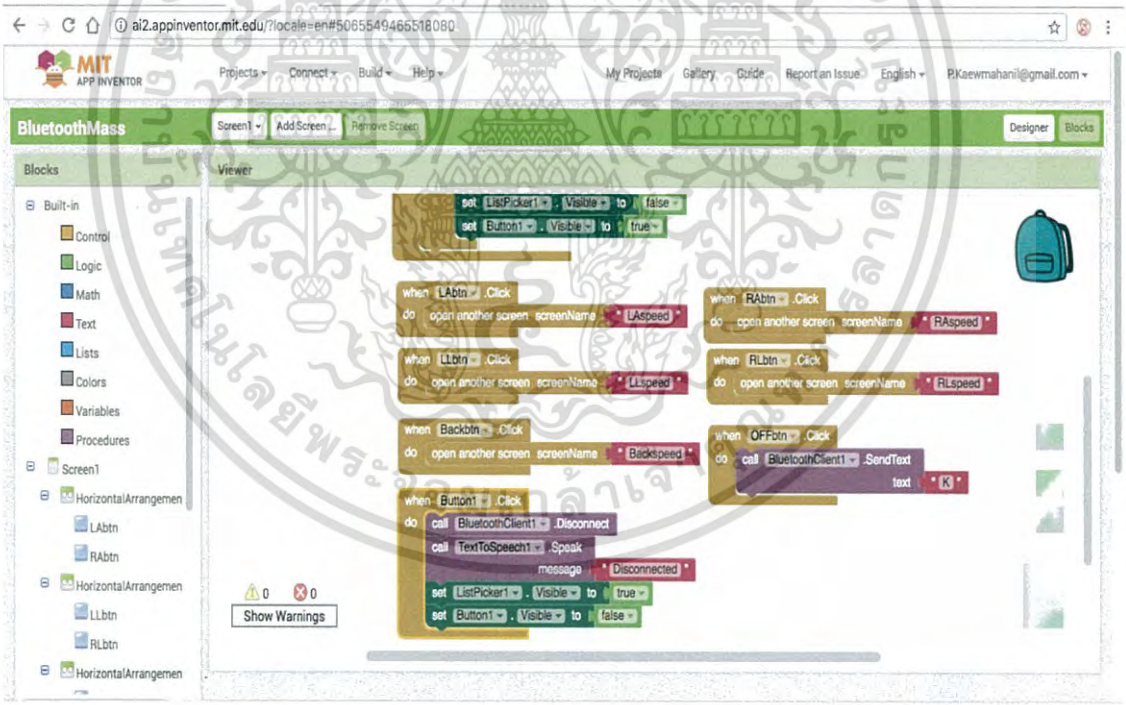
รูปที่ 2.38 App Inventor logo

ที่มา : http://appinventor.mit.edu/explore/blogs/karen/2017/08/about.html



รูปที่ 2.39 หน้าต่างออกแบบ UI

ที่มา : https://goo.gl/iiEYST



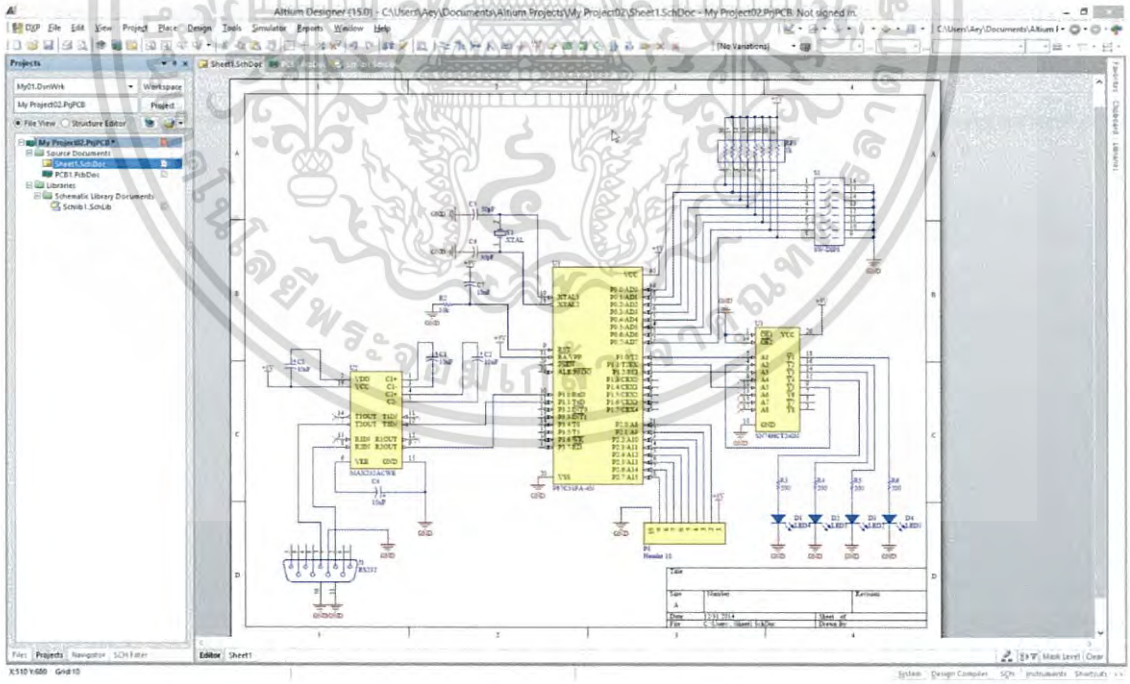
รูปที่ 2.40 หน้าต่างออกแบบคำสั่ง Block

ที่มา : https://goo.gl/iiEYST

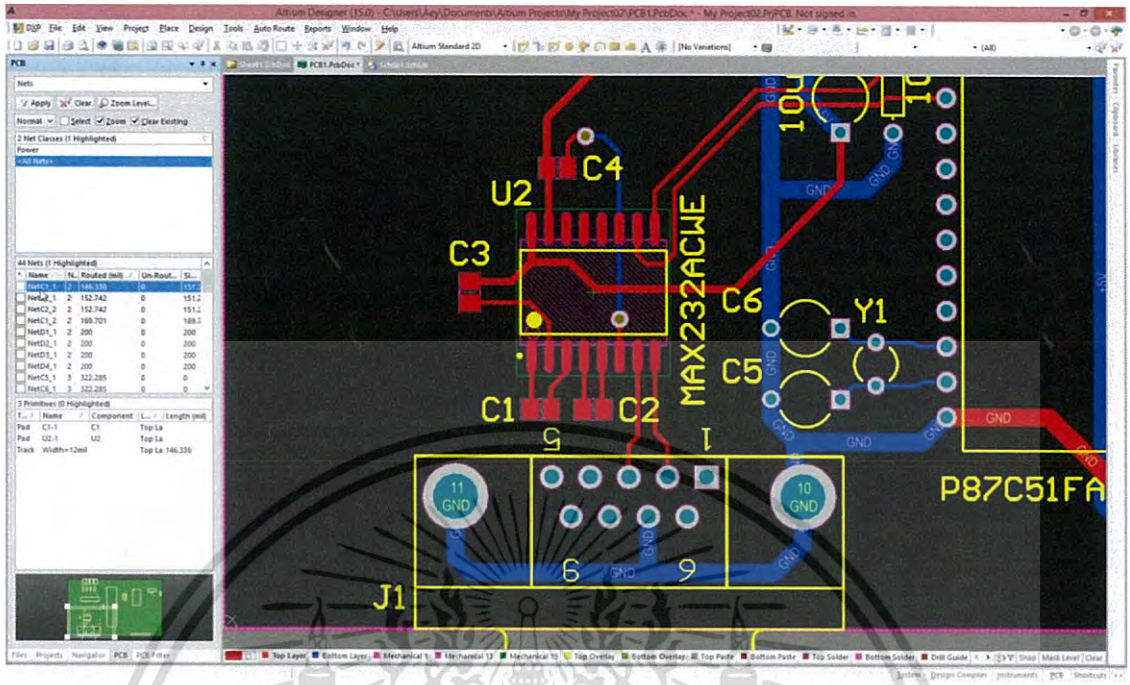
2.17) โปรแกรม Altium Designer

อุปกรณ์อิเล็กทรอนิกส์ต่าง ๆ ที่ใช้งานกันทุก ๆ วัน เช่น โทรทัศน์ วิทยุ คอมพิวเตอร์ เป็นต้น มีการวิจัยและการพัฒนาอย่างมากเพื่อให้ใช้งานได้เต็มที่แต่การวิจัยและการพัฒนาทางด้านอิเล็กทรอนิกส์นั้นจะสำเร็จไปไม่ได้ถ้าขาดสิ่งเล็กๆสิ่งนี้ คือ การออกแบบแผ่นวงจรอิเล็กทรอนิกส์(Print Circuit Board) เนื่องจากแผ่นวงจรอิเล็กทรอนิกส์เป็นศูนย์รวมและเชื่อมต่อกันของชิ้นส่วนต่างๆที่ใช้ในการขับเคลื่อนของอุปกรณ์อิเล็กทรอนิกส์ที่ใช้งานกันอยู่ทุกวัน ซึ่งต้องอาศัยโปรแกรมในการออกแบบ เพื่อสะดวกและรวดเร็ว

รูปที่ 2.41 Altium Designer logo
ที่มา : http://www.altium.com



รูปที่ 2.42 หน้าต่างในส่วนออกแบบ Schematic
ที่มา : http://pcbengineering.lnwshop.com/product/23/altium-designer-15-fundamental



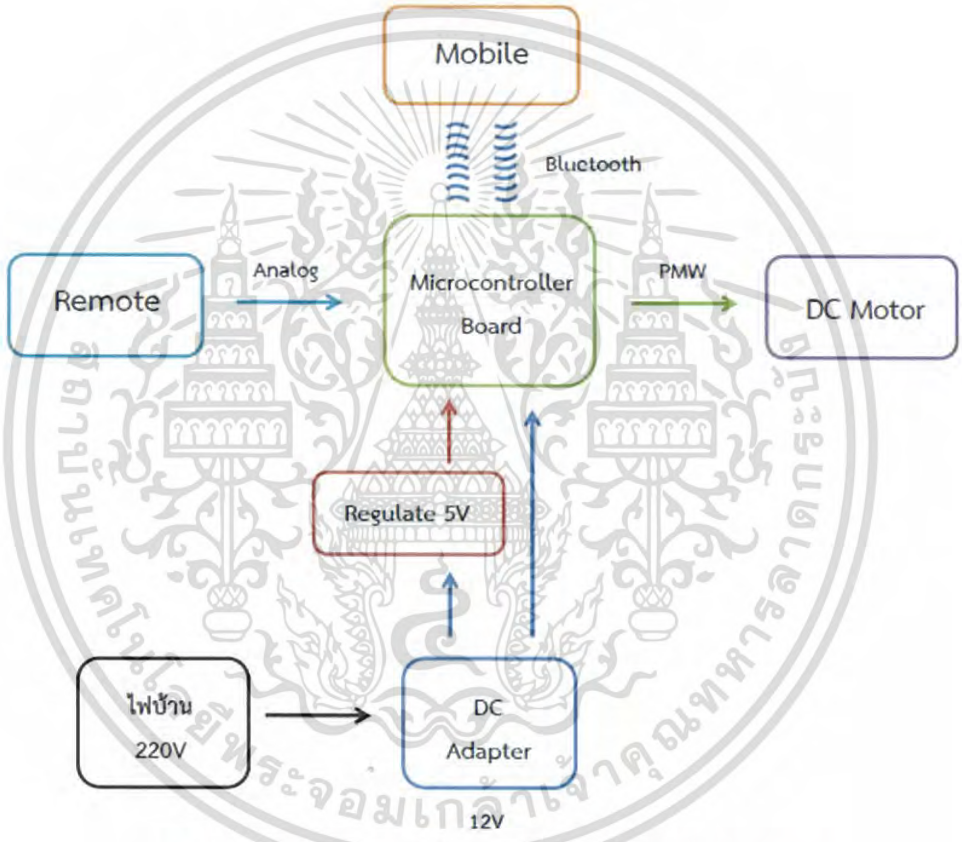
รูปที่ 2.43 หน้าต่างในส่วนออกแบบ PCB Layout

ที่มา : http://pcbengineering.lnwshop.com/product/23/altium-designer-15-fundamental

บทที่ 3
วิธีการดำเนินงาน

3.1) ออกแบบการทำงานของระบบทั้งหมด

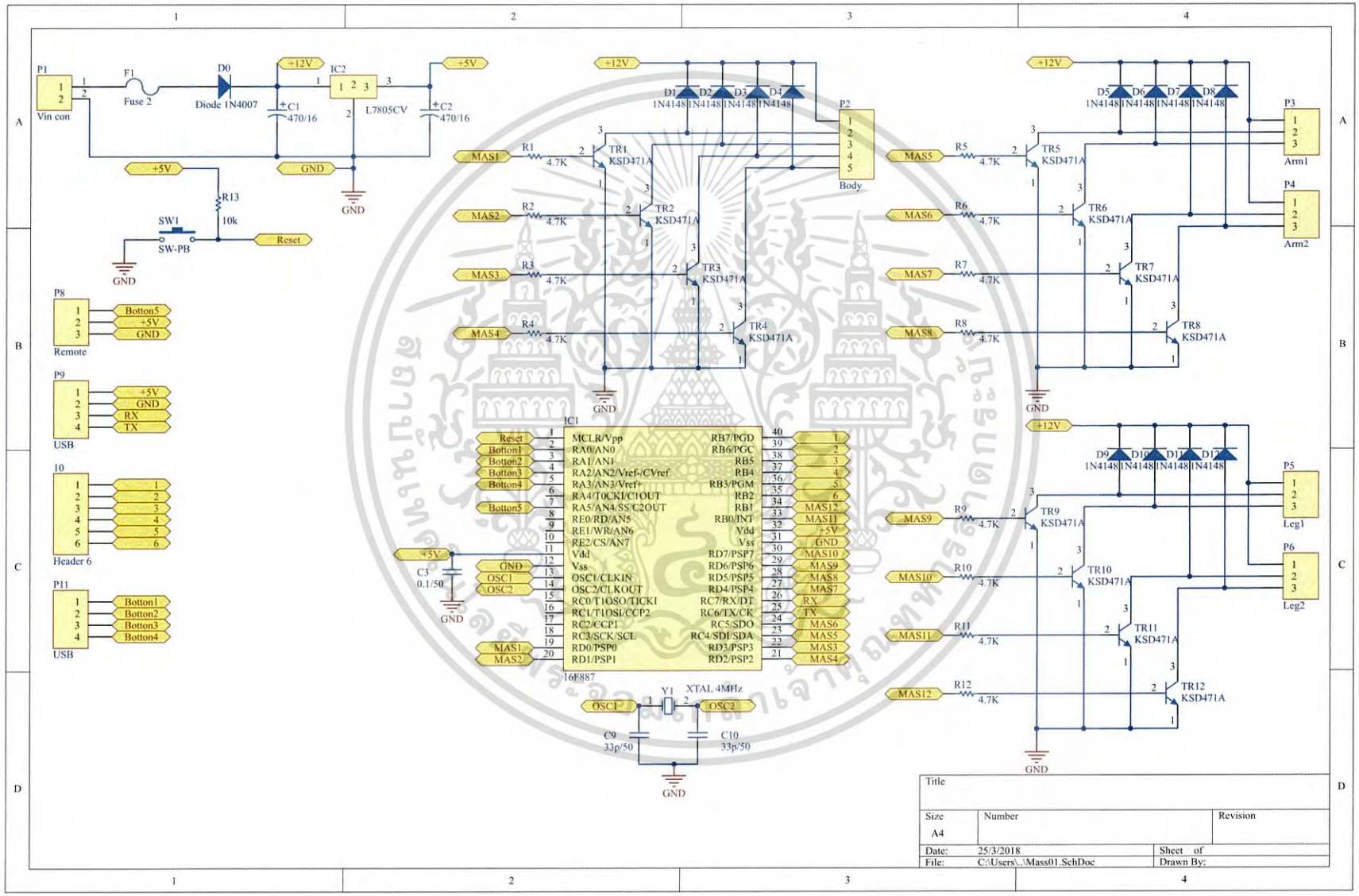
จากที่ได้กล่าวมาในบทที่ 2 ต้องออกแบบวงจรและโปรแกรม สำหรับการทำงานในการนับเวลา, สร้างสัญญาณ PWM, โปรแกรมสำหรับแปลงค่าสัญญาณอนาลอกเป็นสัญญาณดิจิตอล, โปรแกรมรับค่าที่ถูกส่งมาจากโทรศัพท์เข้าบูลทูธ ซึ่งสามารถสรุปขั้นตอนการทำงานได้ดังนี้



รูปที่ 3.1 Flowchart แสดงขั้นตอนการออกแบบระบบการทำงาน

จาก Flowchart สามารถแบ่งการอธิบายเป็นส่วน ๆ ได้ดังนี้

3.1.1) Microcontroller Board จะใช้ไมโครคอนโทรลเลอร์เบอร์ 16F887 ในส่วนนี้จะเป็นการออกแบบด้านฮาร์ดแวร์และซอฟแวร์ร่วมกัน ซึ่งเป็นส่วนสำคัญของการทำงาน มีหน้าที่ในการรับค่าสัญญาณอนาลอกเพื่อนำไปประมวลผลเป็นสัญญาณดิจิตอล, สร้างสัญญาณ PWM เพื่อส่งไปควบคุมระดับความเร็วรอบของ DC motor รวมถึงรับค่าที่ถูกส่งมาจากโทรศัพท์ผ่านสัญญาณบูลทูธ



รูปที่ 3.2 แบบร่างวงจรของ Microcontroller Board

ลำดับ	รายการ	จำนวน
1	PIC16F887	1
2	IC Regulator LM7805CV	1
3	Diode 1N4007	1
4	Diode 1N4148	12
5	Transistor KSD471A	12
6	Resistor 0 Ohm	4
7	Resistor 4.7k Ohm	12
8	Resistor 10k Ohm	1
9	Capacitor 0.1 uF 50V	1
10	Capacitor 470 uF 25V	2
11	Capacitor 33 pF 50V	2
12	Fuse 3 Amp	1
13	Push button switch	1
14	Crystal 4 MHz	1
15	Socket 40 pins	1
16	Connector 2 pins	1
17	Connector 3 pins	5
18	Connector 5 pins	1

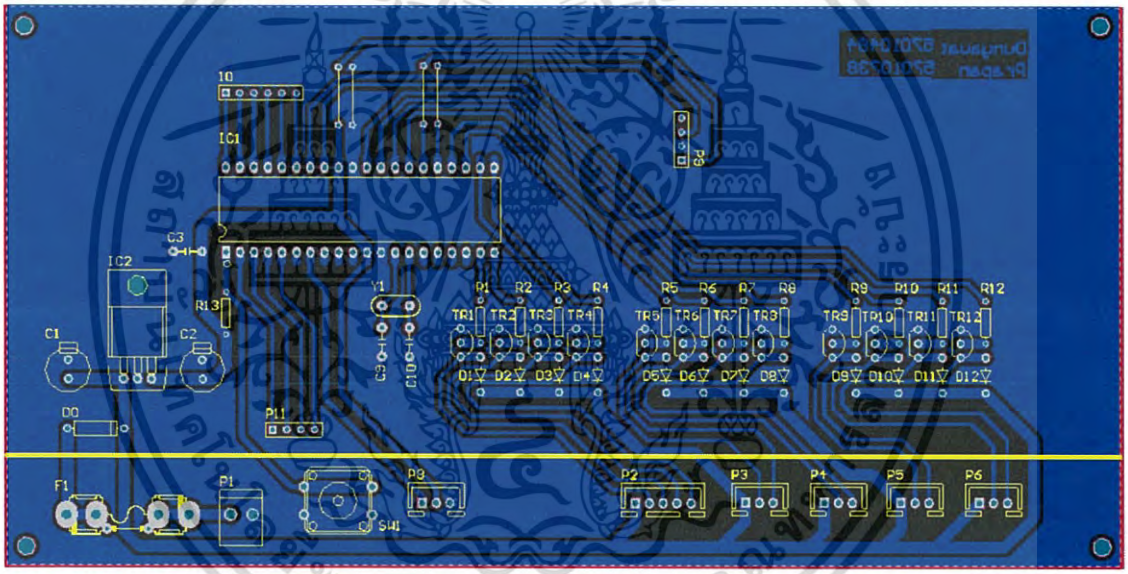
ตารางที่ 3.1 รายการอะไหล่ที่ใช้ใน Microcontroller Board

ไฟเลี้ยงวงจรจะแบ่งเป็น 2 ส่วน คือ

- ส่วนไฟเลี้ยงไมโครคอนโทรลเลอร์ จะเป็นแรงดันไฟตรงขนาด 5 โวลท์ ซึ่งถูกลดระดับลงมาจากแรงดันไฟตรงขนาด 12 โวลท์ โดย IC Regulator LM7805 รวมทั้งยังส่งไปที่รีโมทและ Bluetooth module HC-06 อีกต่อหนึ่งด้วย

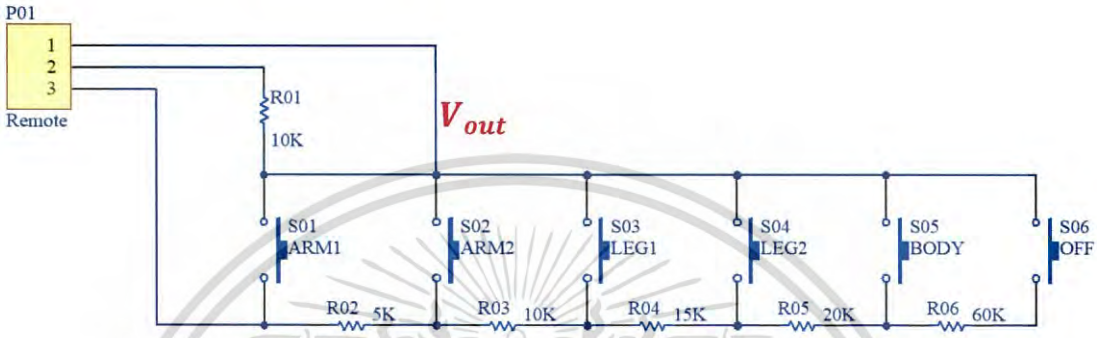
- ส่วนไฟเลี้ยง DC motor ซึ่ง DC motor จะทำงานเต็มพิกัดที่แรงดันไฟตรงขนาด 12 โวลท์

ในส่วนของทรานซิสเตอร์เบอร์ KSD471A จะมีหน้าที่เป็นสวิทช์ซึ่งจะทำงานก็ต่อเมื่อได้รับสัญญาณ PWM ที่ส่งมาจากไมโครคอนโทรลเลอร์เท่านั้น โดยวงจรทรานซิสเตอร์นี้จะถูกต่อเข้ากับ DC motor



รูปที่ 3.3 PCB layout Microcontroller Board

3.1.2) Remote ในส่วนนี้เป็นส่วนที่ใช้สำหรับควบคุมการส่งค่าไปยัง Microcontroller Board โดยหลักการทำงานจะเป็นหลักการแบ่งแรงดัน เมื่อสวิทช์แต่ละปุ่มถูกกด ค่าแรงดันจะมีค่าที่ต่างกันไล่ระดับกันไปตั้งแต่ 0-5 โวลท์ ซึ่งเป็นค่าสัญญาณอนาลอก โดยค่านี้จะส่งไปยังไมโครคอนโทรลเลอร์เพื่อประมวลผล



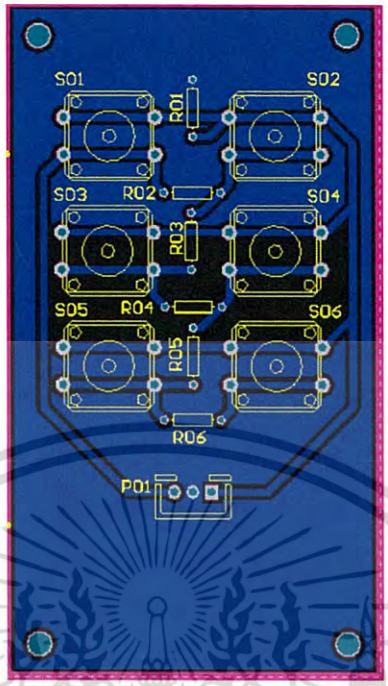
รูปที่ 3.4 แบบร่างวงจรของ Remote

สมการในการคำนวณแรงดัน : $$V_{out} = \frac{R_x}{R_{01}+R_x} \times V_{in} \quad (3.1)$$

เมื่อ $V_{in} = 5\ V$: $R_{01} = 10\ k\Omega$: R_x = ค่าความต้านทานรวมเมื่อกดสวิทช์

ลำดับ	รายการ	จำนวน
1	Resistor 5k Ohm	1
2	Resistor 10k Ohm	2
3	Resistor 15k Ohm	1
4	Resistor 20k Ohm	1
5	Resistor 60k Ohm	1
6	Push button switch	6

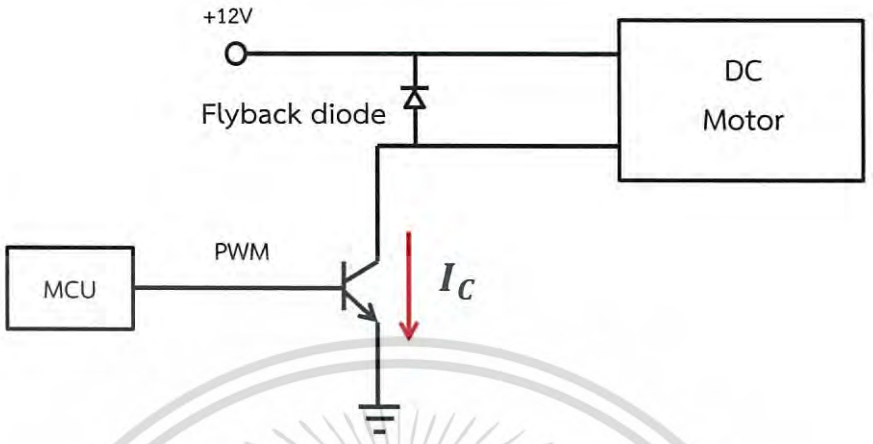
ตารางที่ 3.2 รายการอะไหล่ที่ใช้ใน Remote



รูปที่ 3.5 PCB layout Remote

3.1.3) DC motor เป็นส่วนที่สำคัญอีกส่วนหนึ่งของระบบ เนื่องจาก DC motor จะกำเนิดแรงสั่นสะเทือนเพื่อส่งไปยังร่างกาย โดย DC motor จะหมุนด้วยความถี่และความเร็วรอบ ซึ่งสามารถกำหนดได้จากเขียนโปรแกรมภาษาซี

รูปที่ 3.6 DC motor



รูปที่ 3.7 แสดงวงจรที่เชื่อมต่อกับ DC motor

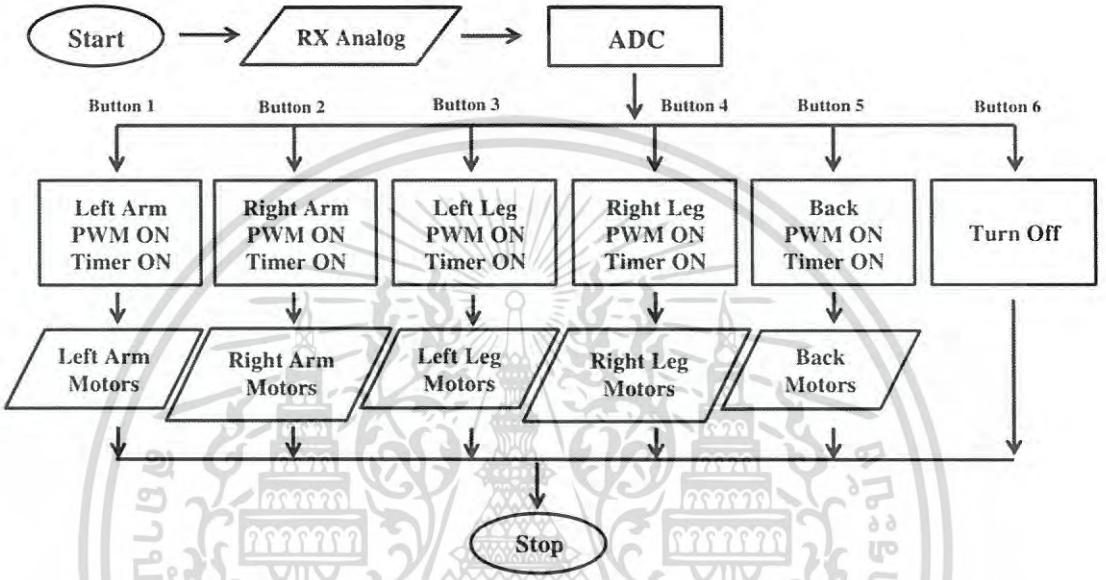
หลักการทำงานของ DC motor คือ เมื่อทรานซิสเตอร์ได้รับสัญญาณ PWM ที่ถูกส่งออกมาจากไมโครคอนโทรลเลอร์แล้ว ทรานซิสเตอร์จะเปลี่ยนสถานะจาก OFF เป็น ON ทำให้กระแส I_C ไหล เกิดกระแสไฟฟ้าไหลครบวงจร DC motor ก็จะหมุนด้วยความถี่และความเร็วรอบคงที่ ตามจังหวะของทรานซิสเตอร์ที่เปลี่ยนสถานะจาก OFF เป็น ON และ ON เป็น OFF ไปจนกว่าเวลาการทำงานจะจบ ซึ่งถูกตั้งเวลาในโปรแกรมภาษาซีในไมโครคอนโทรลเลอร์ไว้ที่ 15 นาที

ในส่วนของ Flyback diode จะใช้เบอร์ 1N4148 ติดตั้งไว้เพื่อป้องกันแรงเคลื่อนไฟฟ้าย้อนกลับ (back EMF) จาก DC motor ขณะชะลอความเร็วให้ไหลเข้าแหล่งจ่าย (DC Bus หรือ DC Link) แทนการตกคร่อมที่อุปกรณ์ (ซึ่งในวงจรคือทรานซิสเตอร์) ช่วยป้องกันอุปกรณ์ภาค Power stage (Power TR หรือ Power MOSFET) ไม่ให้เกิด breakdown โดยแรงเคลื่อนไฟฟ้าย้อนกลับนี้จะเกิดจากขดลวดที่อยู่ในมอเตอร์ซึ่งเมื่อมีกระแสไฟฟ้าไหลผ่าน ขดลวดนั้นจะทำให้เกิดการสะสมพลังงาน ซึ่งหลักการของตัวเหนี่ยวนำ (inductor) ที่แสดงไว้เป็นสมการเชิงอนุพันธ์นั้นระบุว่า ตัวเหนี่ยวนำนั้นไม่สามารถจะมีการเปลี่ยนแปลงของกระแสไฟฟ้าที่ไหลผ่านได้อย่างฉับพลัน ดังนั้นนี่จึงเป็นสาเหตุที่จะทำให้เกิดปัญหาขึ้น

สมการเชิงอนุพันธ์ของขดลวดเหนี่ยวนำ : $V(t) = L\frac{di}{dt}$ (3.2)

3.2) ออกแบบโปรแกรมภาษาซีสำหรับไมโครคอนโทรลเลอร์

ในส่วนนี้จะเป็นการออกแบบซอฟแวร์เพียงอย่างเดียว แต่มีความสำคัญมากต่อการทำงานของระบบ ซึ่งโปรแกรมนั้นจะทำหน้าที่ประมวลผล แปลงค่าสัญญาณ กำเนิดสัญญาณ ตั้งเวลาการทำงาน ซึ่งครอบคลุมฟังก์ชั่นการทำงานทั้งหมด โดยเริ่มต้นต้องออกแบบ Flowchart การทำงานของระบบก่อนได้ดังนี้



รูปที่ 3.8 Flowchart แสดงกระบวนการทำงานของโปรแกรม

เมื่อออกแบบ Flowchart เสร็จสิ้น ต่อไปเป็นขั้นตอนของการเขียนโปรแกรมภาษาซี ซึ่งใช้โปรแกรม MPLAB IDE ในการเขียน ซึ่งจะแบ่งเป็นส่วนสำคัญ ๆ ดังนี้

3.2.1) ส่วนของการรับและแปลงค่าสัญญาณอนาลอกเป็นดิจิตอล

โดยค่าอนาลอกที่ได้มานั้นจะมาจากรีโมท เมื่อไมโครคอนโทรลเลอร์ได้รับค่าอนาลอกมาแล้วจะทำการแปลงข้อมูลเป็นดิจิตอล ซึ่งใช้วิธีการ Successive Approximation Register (SAR) โดยความละเอียดในการแปลงอยู่ที่ 10 บิท และความถี่ของการ Sampling อยู่ที่ 500 kHz

สมการคำนวณค่า ADC ที่ถูกแปลง : $ADC\ result = \frac{V_{in} \times 1023}{V_{ref}}$ (3.3)

เมื่อ $V_{ref} = 5\ V$

<u>ตัวอย่าง</u> $ADC\ result = \frac{4.20 \times 1023}{5} = 859.32 \text{ or } 860$

เมื่อได้ค่าไมโครคอนโทรลเลอร์แปลงค่าอนาลอกเป็นดิจิตอลเสร็จเรียบร้อยแล้วจะนำไปเปรียบกับค่าที่คำนวณได้จากสมการที่ (3.3) ซึ่งคำนวณและเขียนโปรแกรมได้ดังนี้

```
#define OFFSET 20
#define REF_KEY_1 25 + OFFSET //0v
#define REF_KEY_2 365 + OFFSET //1.79v
#define REF_KEY_3 621 + OFFSET //3.04v
#define REF_KEY_4 771 + OFFSET //3.77v
#define REF_KEY_5 860 + OFFSET // 4.2v
#define REF_KEY_6 937 + OFFSET // 4.58v
```

ต่อไปจะเป็นโปรแกรมสำหรับการแปลงค่าอนาลอกเป็นดิจิตอล

```
unsigned int analog(unsigned char channel){
    unsigned int value = 0;
    if(channel >= 0 && channel <= 13)
    {       //CHS<3:0>: Analog Channel Select bits
            CHS3 = (channel & 0x08) == 0x08 ;
            CHS2 = (channel & 0x04) == 0x04 ;
            CHS1 = (channel & 0x02) == 0x02 ;
            CHS0 = (channel & 0x01) == 0x01 ;
            GODONE = 1;             // Start conversion
            while(GODONE);          // Wait until conversion success
            value = (ADRESH<<2)+(ADRESL>>6);   // Get ADC value      }
    return value ;}
```

```
int fn_read_key(void)
{       read_adc = analog(4); //read PIN AN0
        __delay_ms(2);
        if(read_adc < REF_KEY_1){ return btnKEY1;}
        if(read_adc < REF_KEY_2){ return btnKEY2;}
        if(read_adc < REF_KEY_3){ return btnKEY3;}
        if(read_adc < REF_KEY_4){ return btnKEY4;}
        if(read_adc < REF_KEY_5){ return btnKEY5;}
        if(read_adc < REF_KEY_6){ return btnKEY6;}
        return btnNONE;}
```

หลังจากแปลงค่าอนาลอกเป็นดิจิตอลแล้วจะได้ค่า ADC result ซึ่งมีค่าระหว่าง 0 – 1023 โดยจะนำมาเปรียบเทียบในฟังก์ชั่นที่แสดงด้านบน

3.2.2) ส่วนของการสร้างสัญญาณ PWM

หลังจากที่ไมโครคอนโทรลเลอร์แปลงค่าสัญญาณอนาลอกเป็นดิจิตอลแล้ว และถูกส่งไปเปรียบเทียบค่าที่ตั้งไว้ จากนั้นไมโครคอนโทรลเลอร์จะเริ่มกำเนิดสัญญาณ PWM เพื่อส่งไปยังทรานซิสเตอร์แล้ว DC motor จะเริ่มทำงาน

เริ่มต้นเขียนโปรแกรมด้วยการตั้งค่า Register ในไมโครคอนโทรลเลอร์ให้เปิดใช้งานฟังก์ชั่น PWM ดังนี้

```
void Initial_timer2(void)
{       T2CKPS1 = 1;
        T2CKPS0 = 0;
        //Set Prescale 1:16
        TMR2ON = 1; //Enable Timer2
        TMR2IF = 0; //A TMR2 to PR2 match occurred.
        TMR2IE = 1; //TMR2 to PR2 Match interrupt Enabled.
        TMR2 = 0;
        PR2 = 150; //max 8bit or 0xff //lengh of time peroid pwm}
```

จากโค้ดสามารถอธิบายส่วนที่สำคัญได้ดังนี้

1) T2CKPS1 = 1; และ T2CKPS0 = 0; คือ การตั้งค่า Pre-scale หรือการหารความถี่สัญญาณนาฬิกา เพื่อให้มีขนาดที่ลดทอนลง ซึ่งง่ายต่อการคำนวณ จะเห็นได้หารลงไป 16

จะได้ $$T_{osc} = \frac{4\ \text{MHz}}{16} = 250\ \text{kHz}$$

2) TMR2 = 0; และ PR2 = 150 คือ การตั้งค่า Period ของ PWM โดยค่าที่สามารถเปลี่ยนแปลงได้คือ PR2 จะอยู่ในช่วง 0 – 255 สามารถคำนวณ Period ได้ตามสมการต่อไปนี้

สมการคำนวณคาบ PWM : $$\text{PWM Period} = [\text{PR2} + 1] \times 4 \times \frac{1}{T_{osc}} \quad (3.4)$$

เมื่อนำค่าที่ได้มาใส่ในสมการที่ (3.4) จะได้ :

$$\text{PWM Period} = [150 + 1] \times 4 \times \frac{1}{250\ kHz}$$

$$\text{PWM Period} = 2.416\ \text{ms}$$

จากนั้นจะเป็นการเขียนโปรแกรมเพื่อกำหนดความกว้างของ Pulse ดังโปรแกรมต่อไปนี้

```
1   if(TMR2IF == 1) // interval every 2.4 ms
2   {TMR2IF = 0;
3           if(flag.EnMotorArm1){
4           if(flag.Button1 == 1){
5           if(flag.settimerMotorArm1On == 1){
6           count_timePeriodMotor_Arm1 ++;
7           if(count_timePeriodMotor_Arm1 >= TIME_PERIOD_PWM){
8           count_timePeriodMotor_Arm1 = 0;
9           MOTOR5 = 1;
10          MOTOR6 = 1;}
11  else if(count_timePeriodMotor_Arm1 >= count.levelPWM_Arm1){
12          MOTOR5 = 0;
13          MOTOR6 = 0;}}
14          else{ MOTOR5 = 0; MOTOR6 = 0;}}}
```

จากโค้ดสามารถอธิบายส่วนที่สำคัญได้ดังนี้

1) บรรทัดที่ 1 เป็นการเช็คว่า Timer2 นับจำนวนครั้งครบเท่ากับค่า PR2 ที่ตั้งไว้ คือ 150 หรือเป็นเวลา 2.4 ms

2) บรรทัดที่ 2 เป็นการรีเซ็ทค่า Timer2 ให้เป็น 0 เพื่อกลับไปนับใหม่

3) บรรทัดที่ 6 เป็นการบวกค่า Counter ไปทีละ 1

4) บรรทัดที่ 7 เป็นการเปรียบเทียบว่าค่า Counter ที่นับไปมีค่าเท่ากับ TIME_PERIOD_PWM แล้ว ซึ่งในที่นี้ตั้งค่าไว้เท่ากับ 10 โดยค่านี้จะส่งผลถึงคาบของ PWM ด้วยจากเดิมที่คำนวณไว้เท่ากับ PWM Period = 2.416 ms จะกลายเป็น 24.16 ms ซึ่งเพิ่มจากเดิม 10 เท่าตัว สาเหตุของการที่ตั้งไว้เท่ากับ 10 นั้นเพื่อให้สามารถกำหนดค่า Duty cycle ง่ายขึ้นและไม่ติดทศนิยม

5) บรรทัดที่ 9,10 เป็นการส่ง Pulse ให้ MOTOR5 และ MOTOR6 (ออกทางขา RC4 และ RC5)

6) บรรทัดที่11 เป็นการเปรียบเทียบค่า Counter ที่กล่าวไปในข้อที่ 3) กับค่า count.levelPWM_Arm1 ซึ่งตั้งค่าเท่ากับ 3 (หรือจะได้ Duty cycle = 30 %)

7) บรรทัดที่ 12,13 เป็นการส่ง LOW (0 V) ออกทางขา RC4 และ RC5

โดยโปรแกรมจะวนการทำงานเช่นนี้ไปจนกว่าเวลาจะหมด

3.2.3) ส่วนของการสร้างจังหวะในการหมุนของมอเตอร์

ซึ่งการหมุนของมอเตอร์นั้นจะไม่หมุนเพียงแค่จังหวะเดียว ในส่วนนี้จะเป็นการเขียนโปรแกรมเพื่อกำหนดจังหวะในการหมุนของมอเตอร์ เช่น หมุน 1 วินาที หยุด 1 วินาที หรือ หมุน 2 วินาที หยุด 1 วินาที เป็นต้น ซึ่งสามารถเขียนโปรแกรมได้ดังนี้

```
1   if(countMass.timerMotorArm1On == 0 ){
2     countMass.timerMotorArm1On = TIME_MOTOR_1_2_ON_SWING;
3     countMass.timesMotorArm1On++;
4           if(countMass.timesMotorArm1On < TIMES_MOTOR_SWING ){
5           flag.settimerMotorArm1On ^= 1; }
6           else if(countMass.timesMotorArm1On <
            TIMES_MOTOR_SWING+TIMES_MOTOR_NOT_SWING ){
7                 flag.settimerMotorArm1On = 1; }
8           else{countMass.timesMotorArm1On = 0;}}
9     else if(countMass.timerMotorArm1On > 0 )
10          {countMass.timerMotorArm1On --;}
```

จากโค้ดสามารถอธิบายส่วนที่สำคัญได้ดังนี้

1) บรรทัดที่ 1 เป็นการตรวจสอบว่าค่า “countMass.timerMotorArm1On” มีค่าเท่ากับ 0 หรือไม่ ถ้ามีจะเข้ามาทำงานในฟังก์ชั่น

2) บรรทัดที่ 2 เป็นการตั้งค่า “countMass.timerMotorArm1On” ให้มีค่าเท่ากับ “TIME_MOTOR_1_2_ON_SWING” ซึ่งตั้งค่าไว้เท่ากับ 1000 หรือ เป็นระยะเวลา 1 วินาที

3) บรรทัดที่ 3 เป็นการบวกค่า “countMass.timesMotorArm1On” ไปทีละ 1

4) บรรทัดที่ 4 เป็นการเปรียบเทียบว่าค่า “countMass.timesMotorArm1On” มีค่าน้อยกว่า “TIMES_MOTOR_SWING” หรือไม่ หากน้อยกว่าจะเข้ามาทำงานในฟังก์ชั่น ซึ่งถูกตั้งค่าไว้เท่ากับ 10

5) บรรทัดที่ 5 เป็นการกลับค่า “flag.settimerMotorArm1On” เช่น จาก 0 เป็น 1 หรือ จาก 1 เป็น 0 ส่งไปยังฟังก์ชั่น “ข้อที่ 2.2.3) ส่วนของการสร้างสัญญาณ PWM” เพื่อใช้ในการ on หรือ off มอเตอร์ โดยเวลา on เท่ากับ 1 วินาที และ off 1 วินาที ไปเรื่อยๆ จนกระทั่งค่าของ“countMass. timesMotorArm1On” จะเท่ากับ 10 ซึ่งไม่เท่ากับเงื่อนไขในบรรทัดที่ 4 แล้วจะไปเปรียบเทียบอีกครั้งในบรรทัดที่ 6

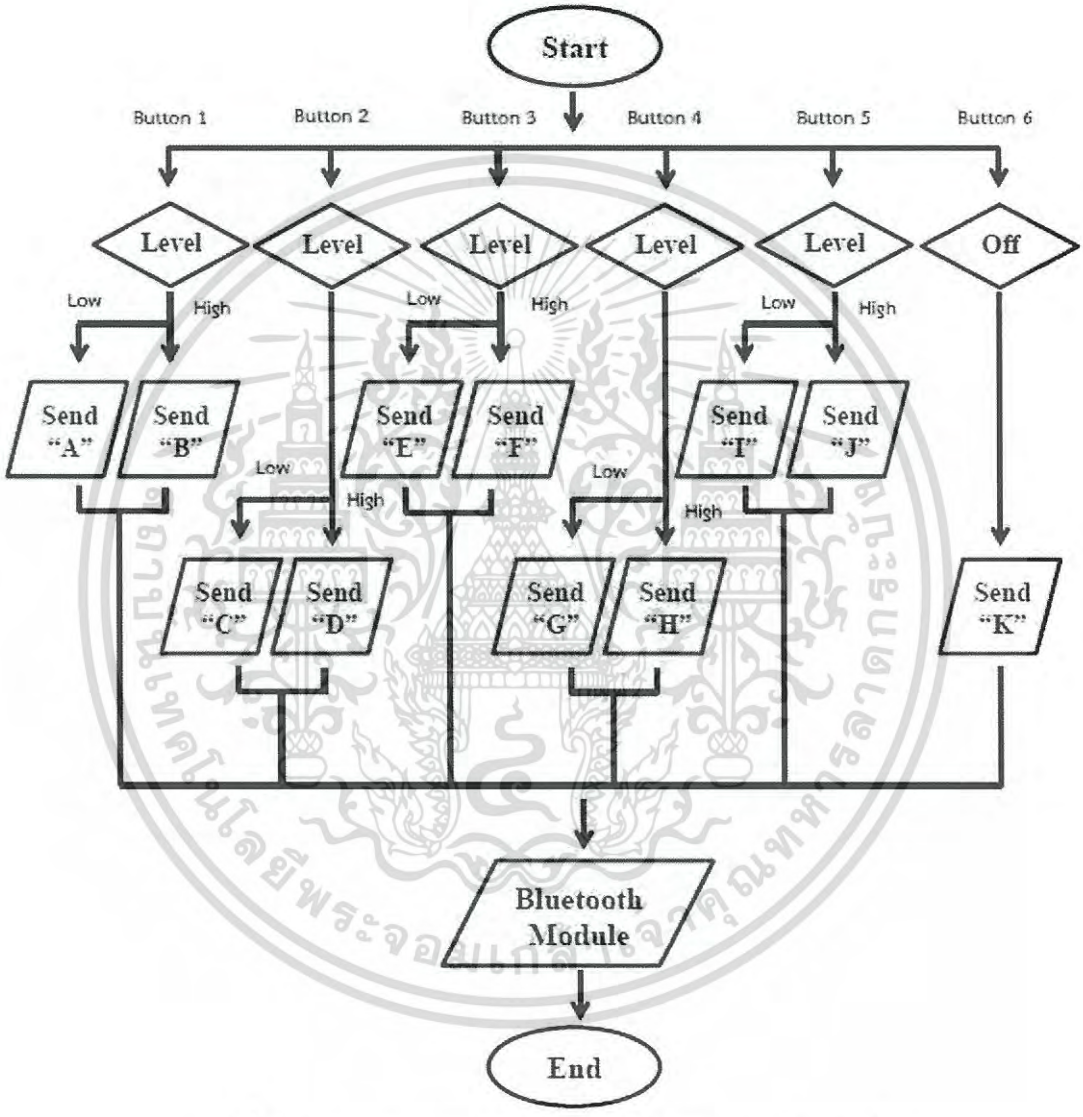
6) บรรทัดที่ 6, 7 เป็นการเปรียบเทียบค่า “countMass.timesMotorArm1On” ว่าน้อยกว่าค่า “TIMES_MOTOR_SWING+TIMES_MOTOR_NOT_SWING” หรือไม่ ซึ่งมีค่าเท่ากับ 10 และ 4 ตามลำดับ โดยค่าที่ได้จะรวมกันเท่ากับ 14 ซึ่งในส่วนนี้จะเป็นการ on ประมาณ 4-5 วินาที

7) บรรทัดที่ 8 เมื่อค่า “countMass.timesMotorArm1On” ไม่ตรงกับเงื่อนไขในบรรทัดที่ 4 และ 6 จะรีเซ็ตค่าให้เป็น 0 เพื่อไปเริ่มต้นใหม่

8) บรรทัดที่ 9, 10 เป็นการตรวจสอบว่าค่า “countMass.timerMotorArm1On” มีค่ามากกว่า 0 หรือไม่ หากมากกว่า 0 จะเข้ามาลบค่าลงเรื่อยๆจนเป็น 0 หรือครบเวลา 1 วินาที โดยโปรแกรมจะวนการทำงานเช่นนี้ไปจนกว่าเวลาจะหมด

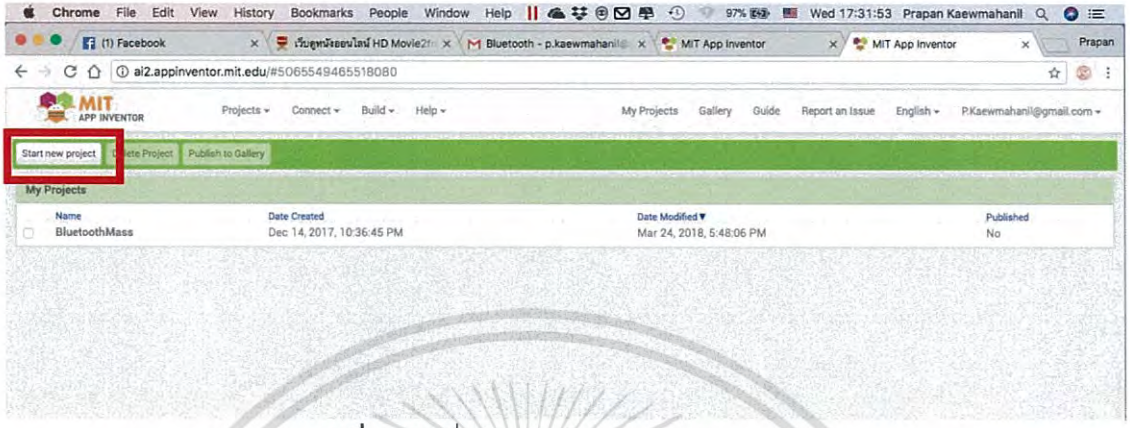
3.2.4) ส่วนของการรับข้อมูลที่ส่งมาจากบูลทูธ

เป็นอีกส่วนเพื่อใช้ในการควบคุมการทำงานของชุดนวดระบบสั่นสะเทือนคือการสั่งการผ่านโทรศัพท์มือถือที่เชื่อมต่อกับสัญญาณบูลทูธ โดยเริ่มต้นได้ทำการออกแบบ Flowchart การทำงานโดยรวมของแอปพลิเคชั่น เพื่อสะดวกในการออกแบบ User Interface ในโทรศัพท์มือถือ ได้ดังรูปต่อไปนี้



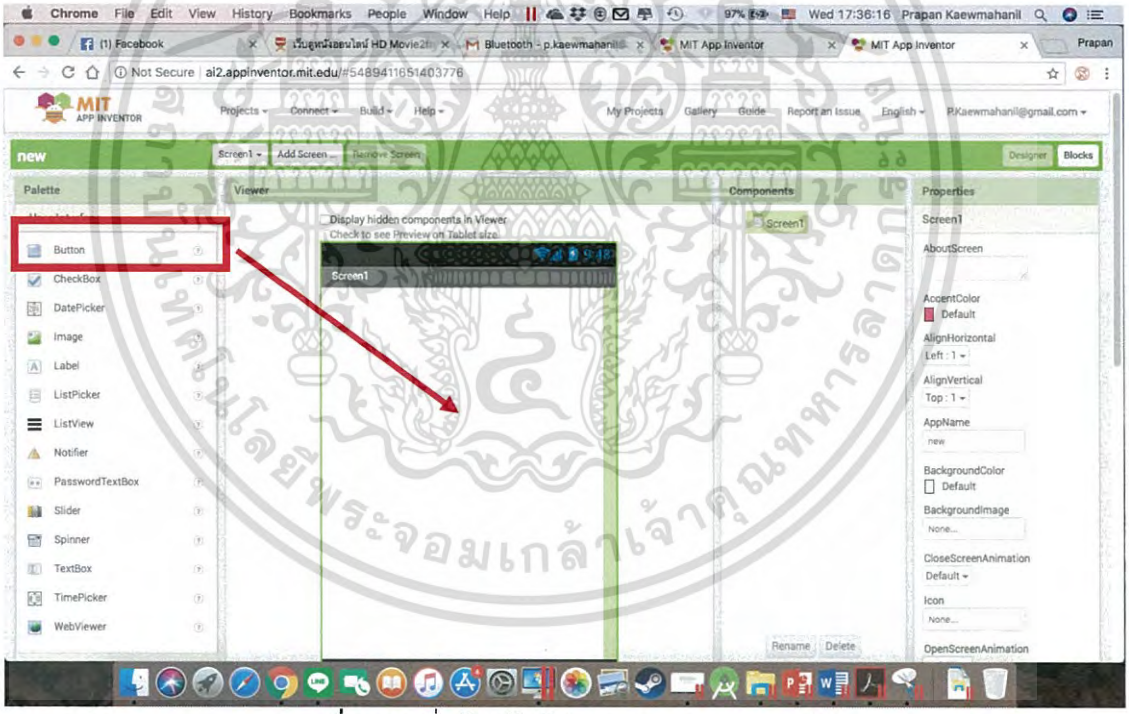
รูปที่ 3.9 Flowchart แสดงกระบวนการทำงานของแอปพลิเคชั่น

เริ่มต้นการออกแบบแอปพลิเคชั่นโดยการเข้าไปที่เว็บไซต์ “ai2.appinventor.mit.edu” จะปรากฏหน้าต่างดังนี้



รูปที่ 3.10 เริ่มต้นสร้างโปรเจค App Inventor

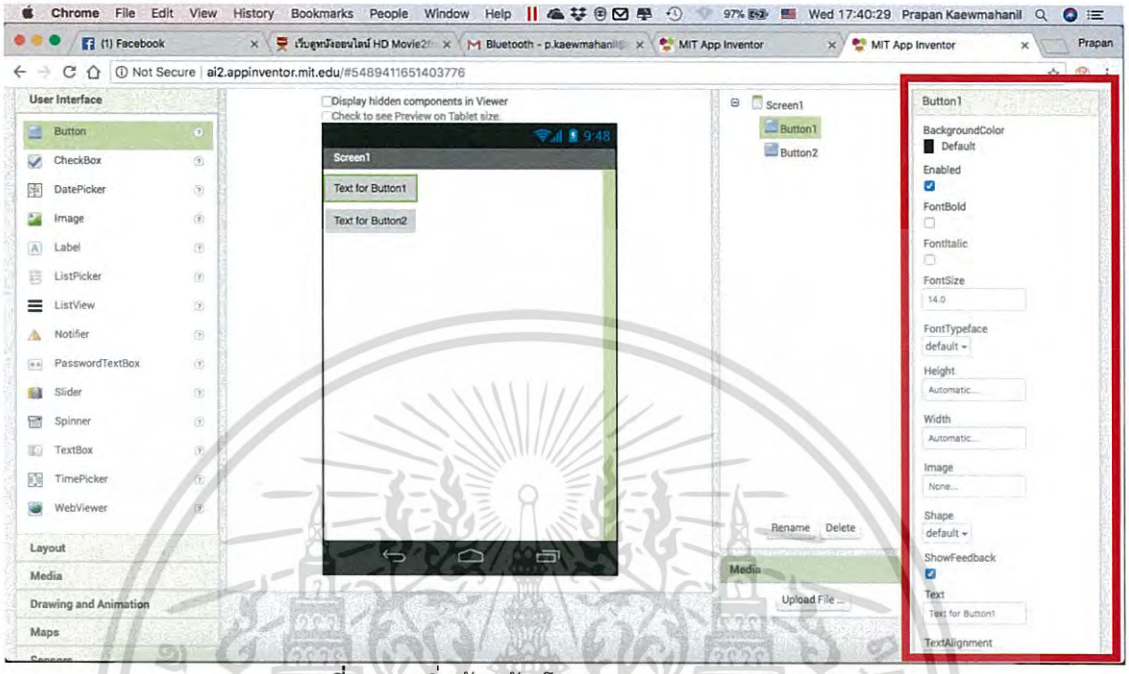
จากนั้นคลิกปุ่ม “ Start new project ” เพื่อสร้างโปรเจคใหม่



รูปที่ 3.11 เริ่มต้นสร้างโปรเจค App Inventor (2)

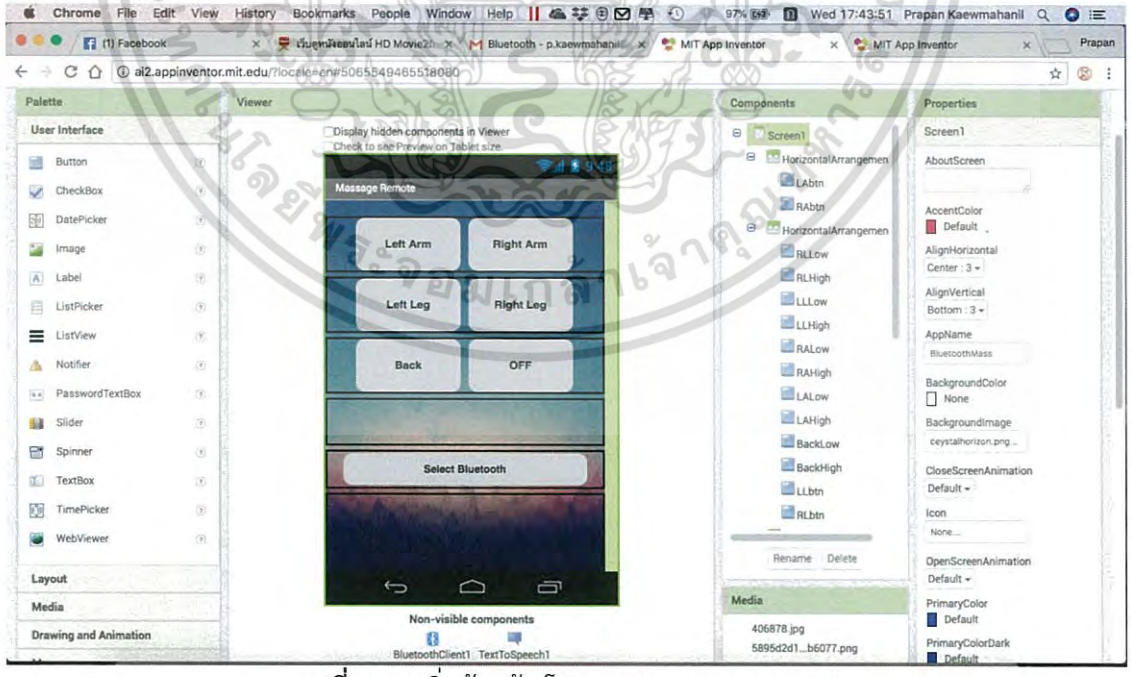
หลังจากสร้างโปรเจคแล้วจะได้หน้าต่างดังรูปที่ 3.6 จากนั้นลากฟังก์ชั่น “Button” ในกรอบสีแดงใส่ลงไปหน้า Screen

หลังจากที่นำ Button ลงมาใส่ใน Screen แล้วจะได้ดังรูปต่อไปนี้ ซึ่งสามารถปรับแต่งรายละเอียดต่าง ๆ ได้ทาง Property ของ Button ในกรอบสีแดงทางขวาของหน้าจอ



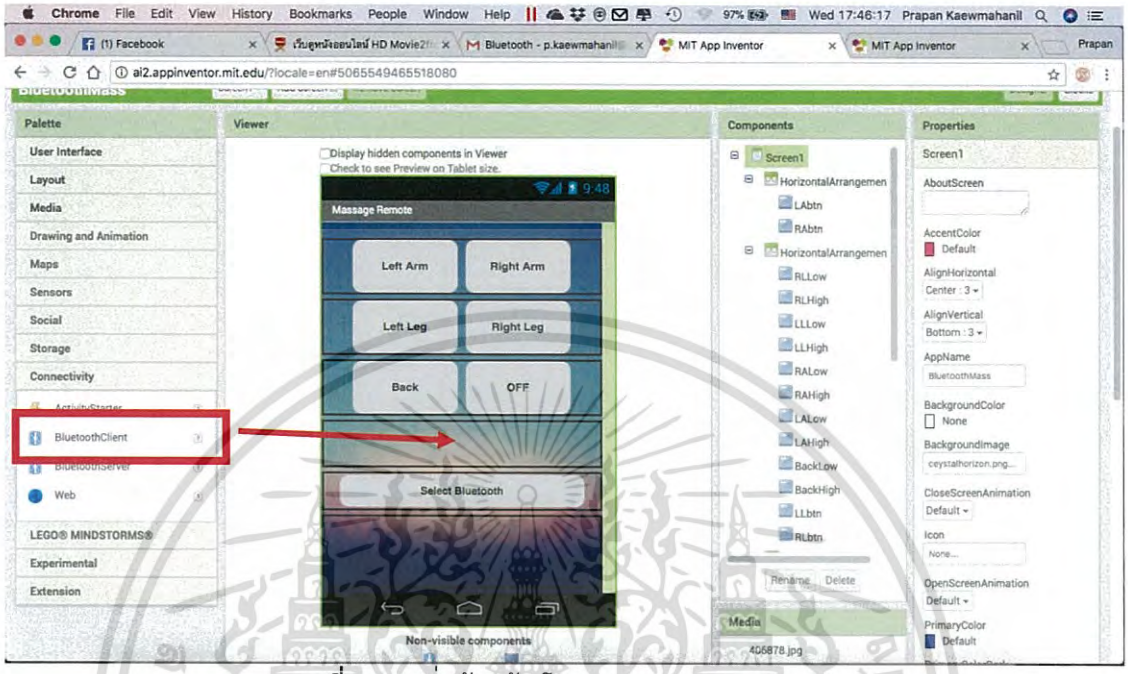
รูปที่ 3.12 เริ่มต้นสร้างโปรเจค App Inventor (3)

เมื่อออกแบบและจัดวาง Button ต่าง ๆ ในการใช้งานและตกแต่งเพื่อความสวยงามจะได้รูปแบบดังต่อไปนี้



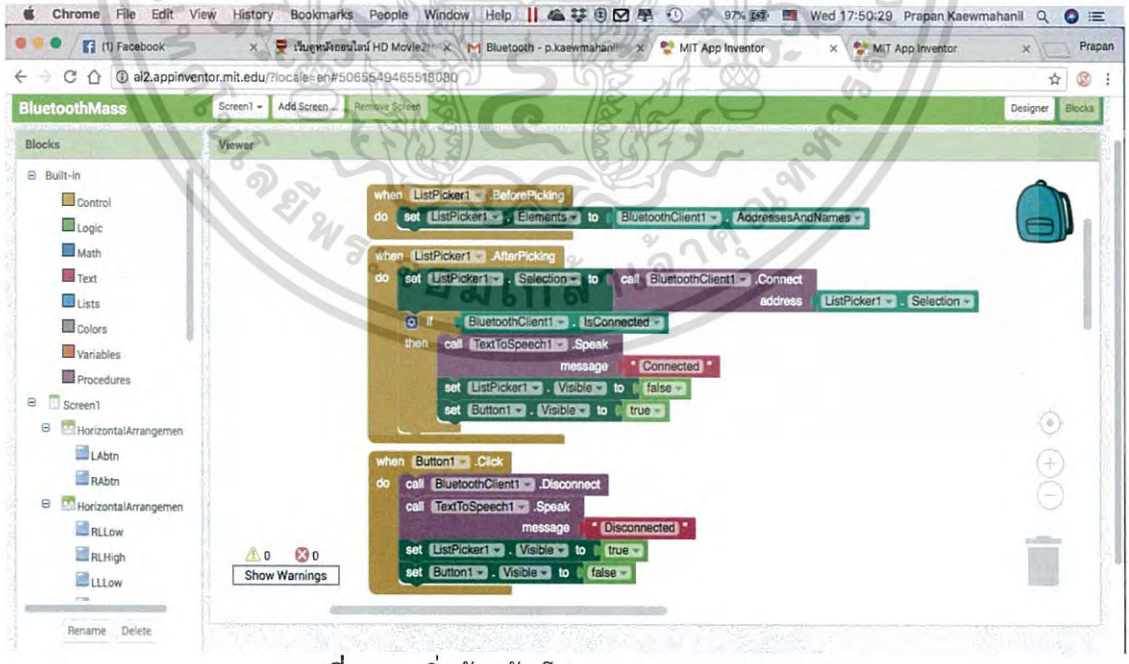
รูปที่ 3.13 เริ่มต้นสร้างโปรเจค App Inventor (4)

ในการที่จะให้แอปพลิเคชั่นเชื่อมต่อกับบูลทูธได้นั้นจำเป็นที่จะต้องมีฟังก์ชั่น BluetoothClient ซึ่งนำมาจาก Tab Connectivity โดยการลากเข้าไปยัง Screen



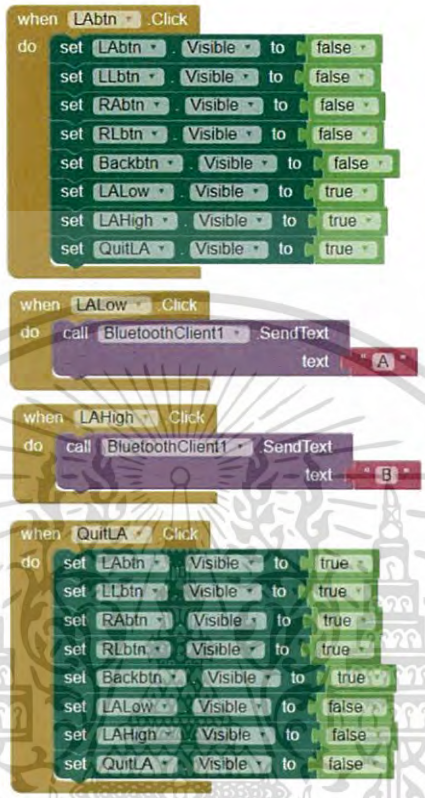
รูปที่ 3.14 เริ่มต้นสร้างโปรเจค App Inventor (5)

จากนั้นไปที่หน้าต่าง Blocks เพื่อทำการออกแบบการทำงานของแอปพลิเคชั่นในการรับ-ส่งข้อมูลและเชื่อมต่อสัญญาณบูลทูธ



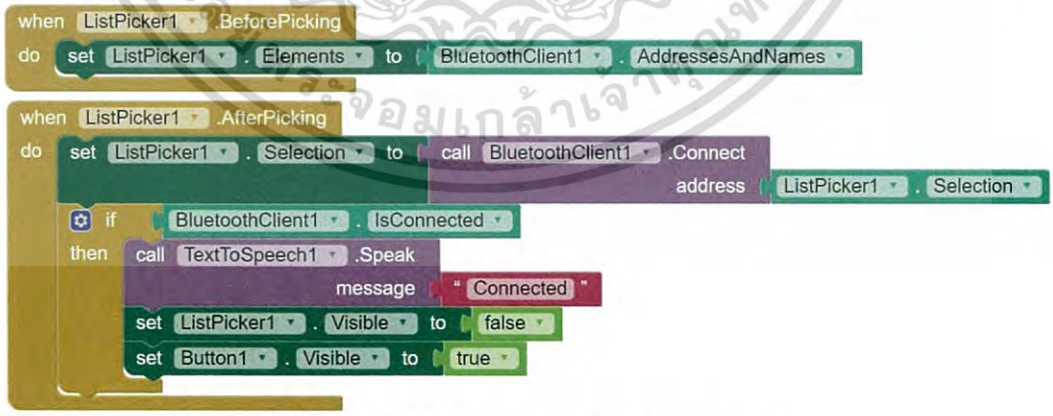
รูปที่ 3.15 เริ่มต้นสร้างโปรเจค App Inventor (6)

แสดง Blocks program ของ button1 จะได้ว่าเมื่อปุ่มถูกกดไป ระบบจะส่งค่า “A” หรือ “B” ไปยังบูลทูธ เพื่อส่งต่อไปที่ไมโครคอนโทรลเลอร์เพื่อประมวลผล

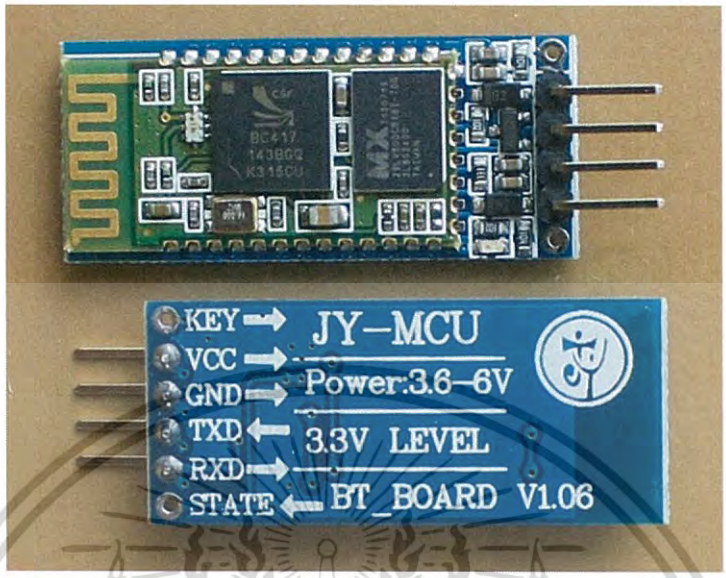


รูปที่ 3.16 เริ่มต้นสร้างโปรเจค App Inventor (7)

รูปต่อไปจะแสดง Blocks program ของฟังก์ชั่นที่ใช้สำหรับการเชื่อมต่อบูลทูธ ซึ่งจะค้นหา Address ของบูลทูธที่จะทำการเชื่อมต่อ เมื่อเชื่อมต่อสำเร็จระบบจะขึ้นคำว่า “connected” บนหน้าจอ



รูปที่ 3.17 เริ่มต้นสร้างโปรเจค App Inventor (8)

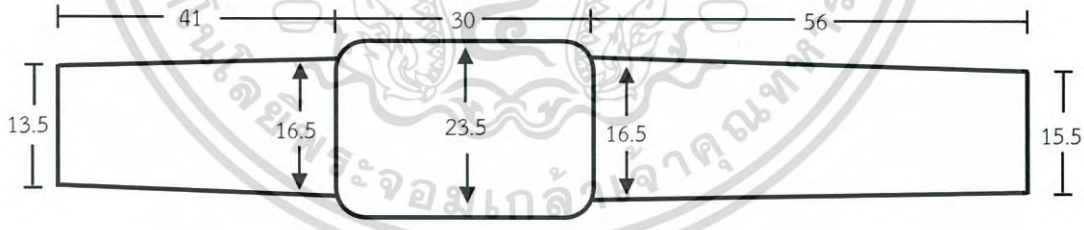


รูปที่ 3.18 Bluetooth Module HC-06

3.3) การออกแบบชุดนวด

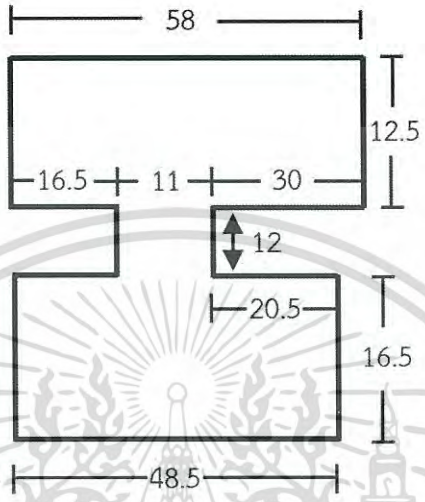
ซึ่งวัตถุดิบที่จะนำมาใช้ในการตัดเย็บชุดจะเป็นหนังเทียม PU และฟองน้ำ โดยแบ่งออกเป็น 5 ส่วนหลักๆ ซึ่งมีหน่วยการวัดเป็นเซนติเมตร ดังนี้

3.3.1) ส่วนหลัง



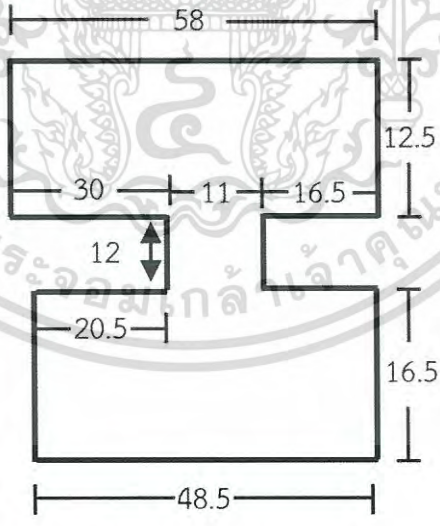
รูปที่ 3.19 โครงสร้างของชุดนวดส่วนหลัง

3.3.2) แขนซ้าย



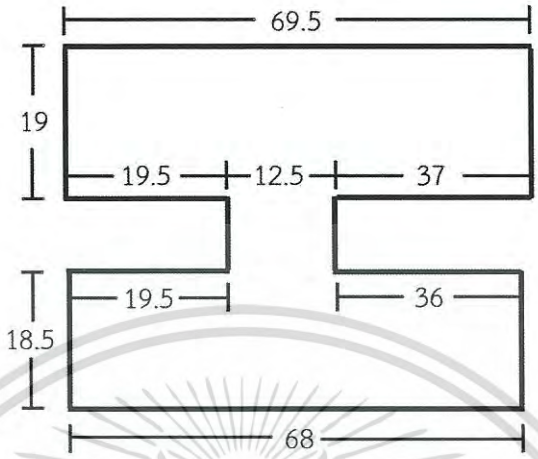
รูปที่ 3.20 โครงสร้างของชุดนวดแขนซ้าย

3.3.3) แขนขวา



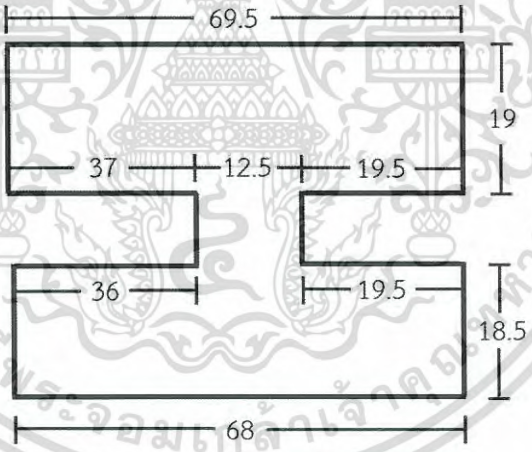
รูปที่ 3.21 โครงสร้างของชุดนวดแขนขวา

3.3.4) ขาซ้าย



รูปที่ 3.22 โครงสร้างของชุดนวดขาซ้าย

3.3.5) ขาขวา

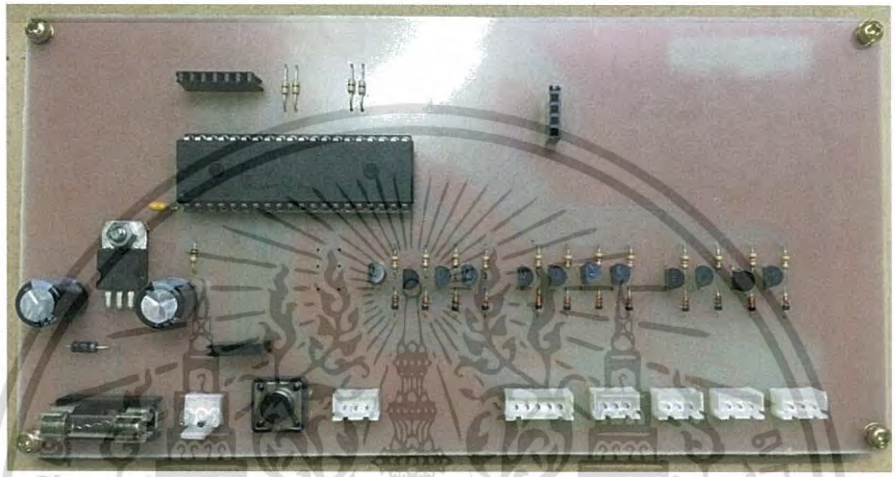


รูปที่ 3.23 โครงสร้างของชุดนวดขาขวา

บทที่ 4
ผลการดำเนินงาน

4.1) ผลการประกอบชิ้นส่วนอะไหล่ลงบน Microcontroller Board

เมื่อนำชิ้นส่วนอะไหล่ในบทที่ 3 ตารางที่ 3.1 มาประกอบลงบน Microcontroller Board ได้ดังรูป



รูปที่ 4.1 Microcontroller Board ที่ประกอบเสร็จสิ้น

4.2) ผลการประกอบชิ้นส่วนอะไหล่ลงบน Remote Board

เมื่อนำชิ้นส่วนอะไหล่ในบทที่ 3 ตารางที่ 3.2 มาประกอบลงบน Remote Board ได้ดังรูป

รูปที่ 4.2 Remote Board ที่ประกอบเสร็จสิ้น

4.2.1) ทดสอบวัดระดับแรงดันเมื่อกดปุ่ม

ทดสอบโดยการป้อนไฟฟ้ากระแสตรงเข้าวงจรรีโมทแล้วทำการวัดหาค่าแรงดัน เพื่อใช้ในการเปรียบกับค่าที่คำนวณได้ในสมการที่ 3.1) ได้ดังนี้

ปุ่มกด	แรงดัน [V]			
	ครั้งที่ 1	ครั้งที่ 2	ครั้งที่ 3	เฉลี่ย
1	0	0	0	0
2	1.79	1.79	1.79	1.79
3	3.03	3.03	3.03	3.03
4	3.76	3.76	3.76	3.76
5	4.19	4.2	4.19	4.19
6	4.57	4.57	4.57	4.57

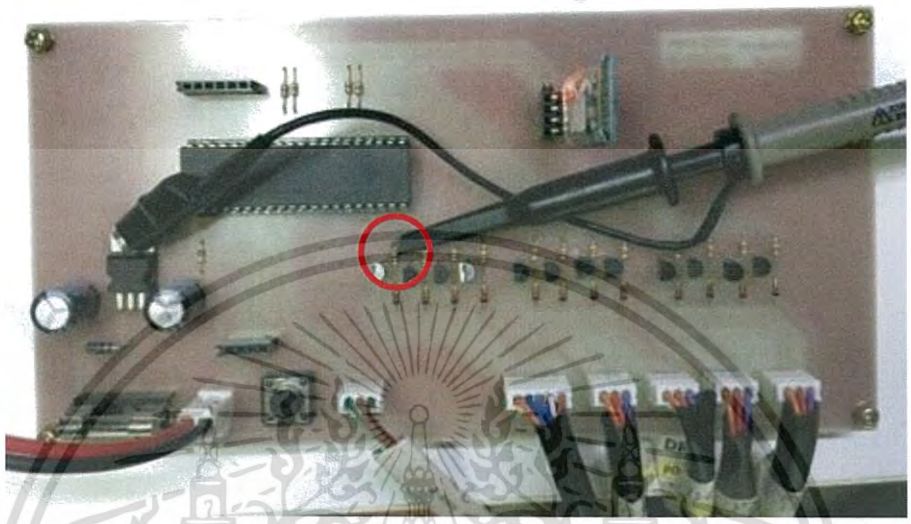
ตารางที่ 4.1 วัดค่าแรงดันเมื่อกดปุ่ม

ปุ่มกด	แรงดัน [V]		
	จากการทดลอง	จากคำนวณ	% ความแตกต่าง
1	0	0	0
2	1.79	1.66	7.81
3	3.03	3	1
4	3.76	3.75	0.26
5	4.19	4.16	0.72
6	4.57	4.58	0.21

ตารางที่ 4.2 เปรียบเทียบค่าที่วัดได้และจากการคำนวณ

4.3) ผลการทดลองสร้างสัญญาณ PWM เพื่อไปควบคุมการทำงาน DC Motor

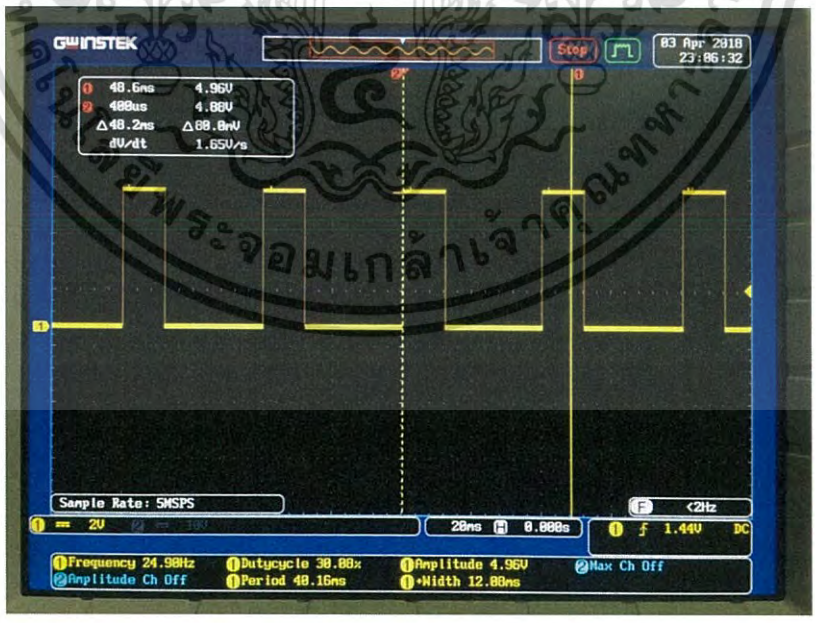
เมื่อประกอบอุปกรณ์ทั้งหมดเรียบร้อยแล้วและทำการเบิร์นโปรแกรมภาษาซีที่เขียนไว้ในโปรแกรม MPLAB IDE ไปยัง PIC16F887 จากนั้นทำการทดสอบวัดสัญญาณ PWM ในตำแหน่งดังรูป



รูปที่ 4.3 ตำแหน่งในการวัดสัญญาณ PWM

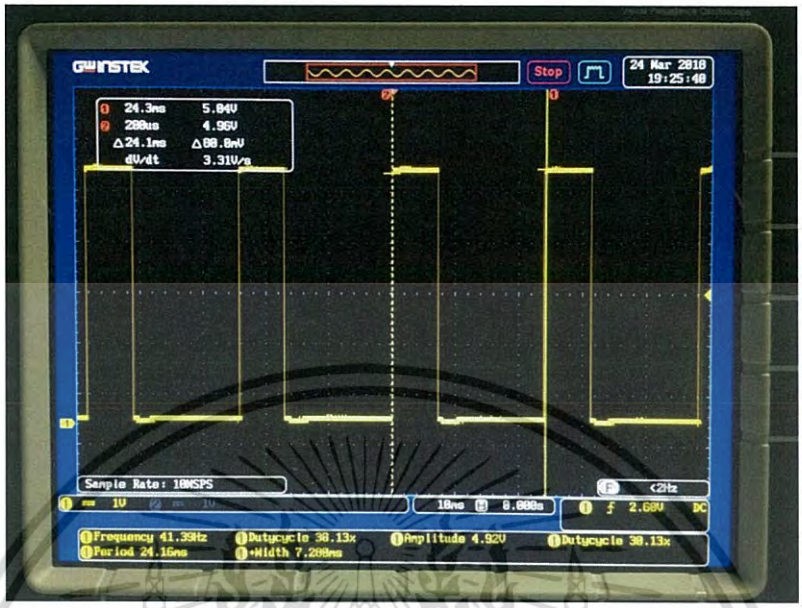
โดยจะแบ่งการกำเนิดสัญญาณ PWM เป็น 3 ความถี่ ดังนี้

4.3.1) ที่ความถี่ 24.90 Hz



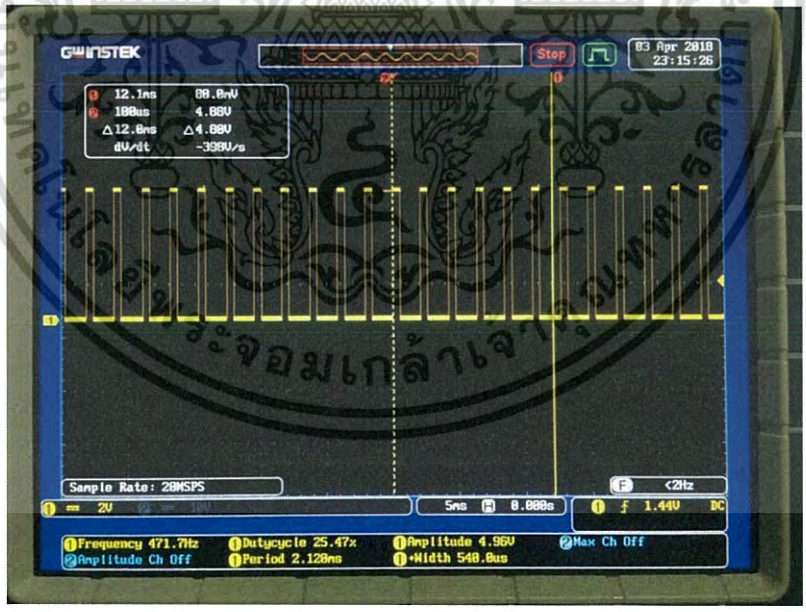
รูปที่ 4.4 สัญญาณ PWM ที่ความถี่ 24.90 Hz

4.3.2) ที่ความถี่ 41.39 Hz



รูปที่ 4.5 สัญญาณ PWM ที่ความถี่ 41.39 Hz

4.3.3) ที่ความถี่ 471.7 Hz



รูปที่ 4.6 สัญญาณ PWM ที่ความถี่ 471.7 Hz

ความถี่ [Hz]	Duty Cycle [%]	Period [ms]	Width [ms]	Voltage [V]
24.9	30	40.16	12.08	1.551
41.39	30	24.16	7.28	1.499
471.7	30	2.12	0.54 [us]	1.259

ตารางที่ 4.3 เปรียบเทียบค่าที่วัดได้ในแต่ละความถี่

จากการทดลองกำเนิดสัญญาณ PWM ที่ความถี่ 24.9 Hz, 41.39 Hz และ 471.7 Hz โดยไม่เปลี่ยนแปลงค่า Duty Cycle พบว่าเมื่อความถี่เพิ่มมากขึ้น ค่าของ Period, Width และ Voltage ที่วัดได้จะมีค่าลดลงตามตารางที่ 4.1

4.4) ผลการทดลองการวัดกระแสที่ผ่าน DC Motor

การทดลองจะเป็นการกำเนิดสัญญาณ PWM ที่ความถี่ 24.9 Hz, 41.39 Hz และ 471.7 Hz และ Duty Cycle ตั้งแต่ 0-100 % ไปยังขา B ของทรานซิสเตอร์ซึ่งทำหน้าที่เป็นสวิตซ์ในการ on/off ของ DC Motor ที่ต่ออนุกรมกับขา C ของทรานซิสเตอร์ซึ่งได้ผลการทดลองดังนี้

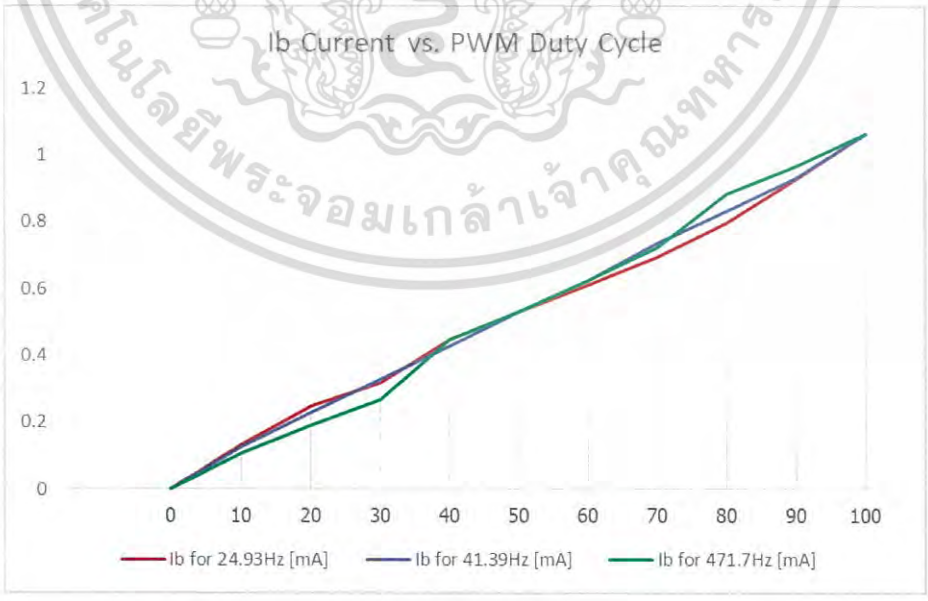
PWM Duty Cycle [%]	Ib for 24.93Hz [mA]	Ib for 41.39Hz [mA]	Ib for 471.7Hz [mA]
0	0	0	0
10	0.132	0.127	0.106
20	0.25	0.228	0.192
30	0.319	0.33	0.268
40	0.447	0.428	0.446
50	0.53	0.532	0.531
60	0.611	0.623	0.623
70	0.694	0.736	0.723
80	0.798	0.832	0.883
90	0.928	0.932	0.968
100	1.064	1.064	1.064

ตารางที่ 4.4 เปรียบเทียบค่ากระแส Ib

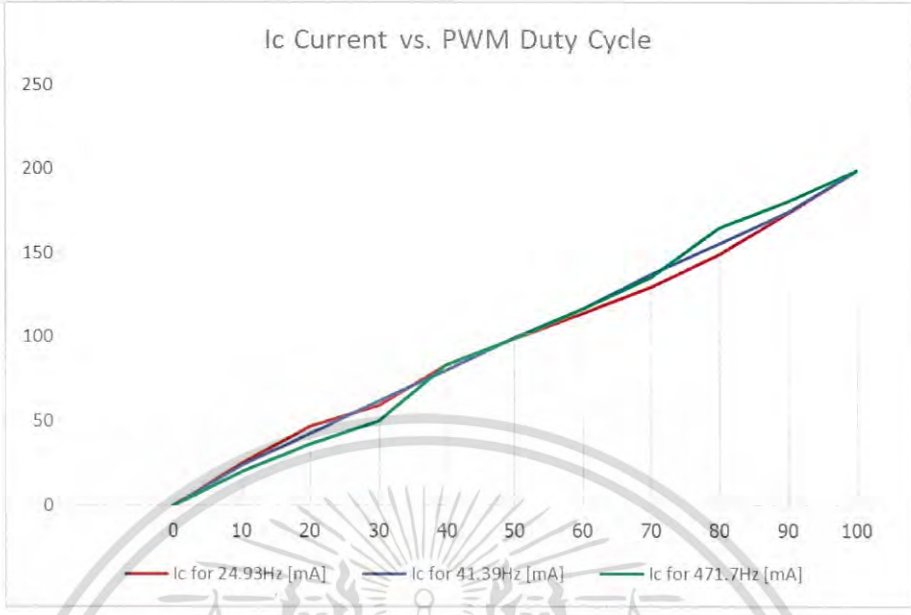
PWM Duty Cycle [%]	Ic for 24.93Hz [mA]	Ic for 41.39Hz [mA]	Ic for 471.7Hz [mA]
0	0	0	0
10	24.684	23.749	19.822
20	46.75	42.636	35.904
30	59.653	61.71	50.116
40	83.589	80.036	83.402
50	99.11	99.484	99.297
60	114.257	116.501	116.501
70	129.778	137.632	135.201
80	149.226	155.584	165.121
90	173.536	174.284	181.016
100	198.968	198.968	198.968

ตารางที่ 4.5 เปรียบเทียบค่ากระแส Ic

จากค่าที่ได้ในตารางที่ 4.2 และ 4.3 สามารถนำมาเขียนกราฟได้ดังนี้



รูปที่ 4.7 กราฟแสดงว่าสัมพันธ์ระหว่าง Ib กับ Duty Cycle



รูปที่ 4.8 กราฟแสดงว่าสัมพันธ์ระหว่าง Ic กับ Duty Cycle

จากการวิเคราะห์กราฟแสดงความสัมพันธ์ระหว่าง Ib กับ duty Cycle ถ้า Duty Cycle เพิ่ม กระแส Ib ที่ได้ก็จะยิ่งมากขึ้น และถ้าความถี่เพิ่มมากขึ้นกระแส Ib ก็จะลดลง

จากการวิเคราะห์กราฟแสดงความสัมพันธ์ระหว่าง Ic กับ duty Cycle ถ้า Duty Cycle เพิ่ม กระแส Ic ที่ได้ก็จะยิ่งมากขึ้น และถ้าความถี่เพิ่มมากขึ้นกระแส Ic ก็จะลดลง โดยกระแส Ic จะมีผลก็ต่อเมื่อกระแส Ib มีค่าเพิ่มมากขึ้นและ หากกระแส Ib มีค่าน้อยลงกระแส Ic ก็จะมีค่ากระแสน้อยลง เช่นกัน

ตามสมการคำนวนค่ากระแสของทรานซิสเตอร์

$$I_c = \beta I_b \tag{4.1}$$

เมื่อ

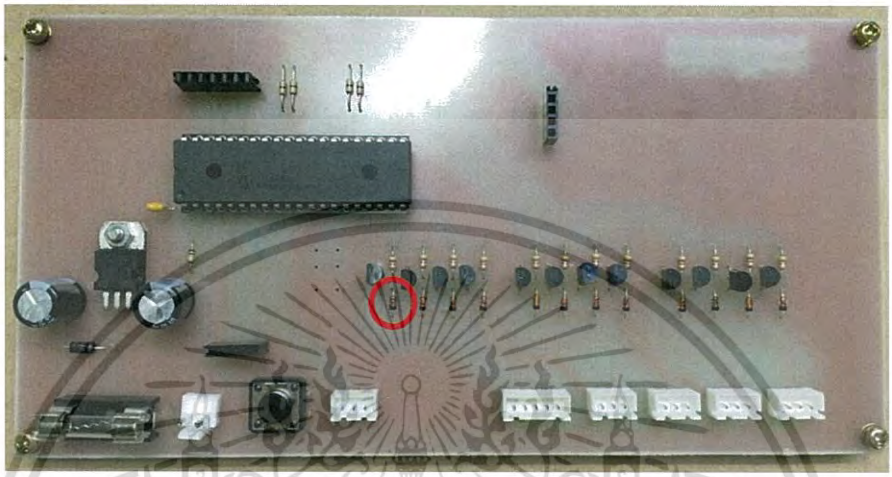
I_c = กระแสที่ไหลผ่านขา C ของทรานซิสเตอร์

I_b = กระแสที่ไหลผ่านขา B ของทรานซิสเตอร์

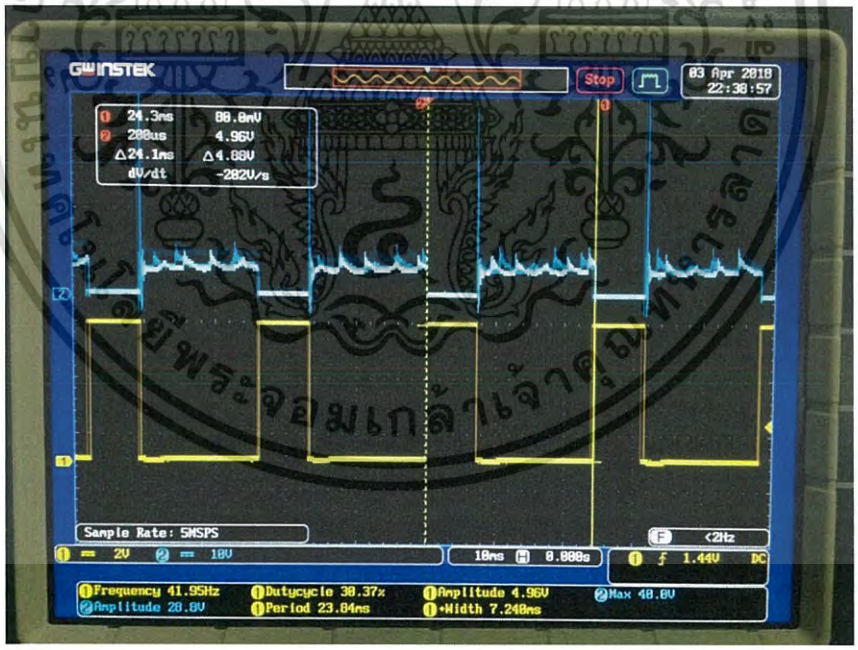
β = DC current gain (h_{FE})

4.5) ผลการทดสอบเปรียบเทียบการใส่และไม่ใส่ไดโอด 1N4148

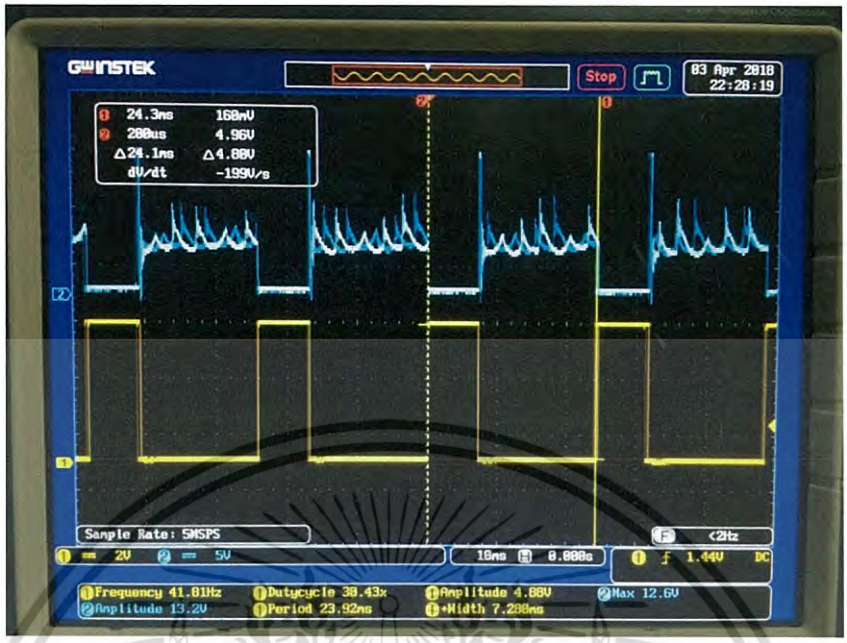
จากที่กล่าวไปในบทที่ 3 หัวข้อ 3.1.3 ได้ทำการทดสอบระหว่างการใส่ไดโอด 1N4148 ขนานกับ DC Motor และไม่ใส่ไดโอดสามารถวัดผลได้จากออสซิโลสโคปบริเวณขาแอโนด(+)ของไดโอดในวงกลมสีแดงดังรูปต่อไปนี้



รูปที่ 4.9 ตำแหน่งการวัดสัญญาณที่ไดโอด 1N4148



รูปที่ 4.10 สัญญาณที่วัดจาก DC Motor เมื่อไม่มีไดโอด

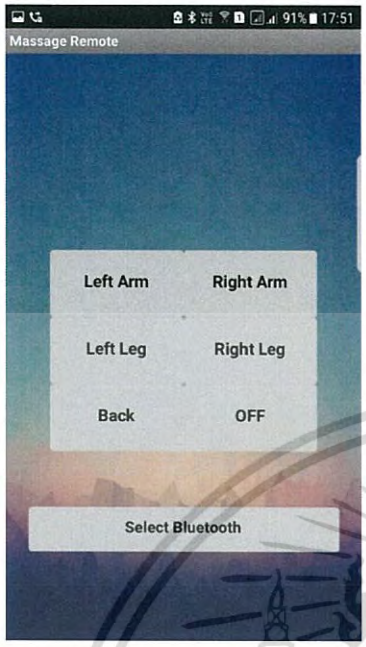


รูปที่ 4.10 สัญญาณที่วัดจาก DC Motor เมื่อมีไดโอด

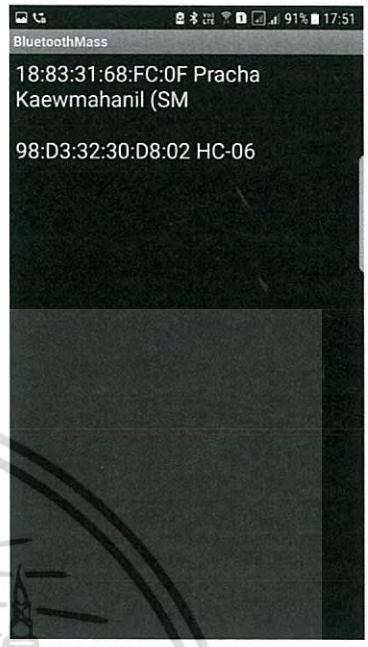
จากการทดสอบป้อนสัญญาณ PWM ที่ความถี่ 41.95 Hz, Period 23.84 ms, Width 7.2 ms, Duty Cycle 30% ซึ่งสัญญาณสีเหลืองจะเป็นสัญญาณ PWM และสัญญาณสีฟ้าจะเป็นสัญญาณจาก DC Motor พบว่าเมื่อไม่มีการใส่ไดโอด 1N4148 จะมีแรงเคลื่อนไฟฟ้าย้อนกลับ (back EMF) จาก DC motor ในขณะที่สัญญาณ PWM มีสถานะเป็น Low สูงถึง 40 V จากรูปที่ 4.9 ซึ่งแรงเคลื่อนไฟฟ้าย้อนกลับนี้จะมีผลกระทบถึงทรานซิสเตอร์ทำให้พังได้เมื่อใช้งานไปในระยะเวลาหนึ่ง แต่ถ้าหากมีการใส่ไดโอด 1N4148 ขนานกับ DC Motor ผลปรากฏว่าแรงเคลื่อนไฟฟ้าลดต่ำลงอย่างมากถึง 68.5% ดังรูปที่ 4.10

4.6) ผลการทดสอบแอปพลิเคชั่น

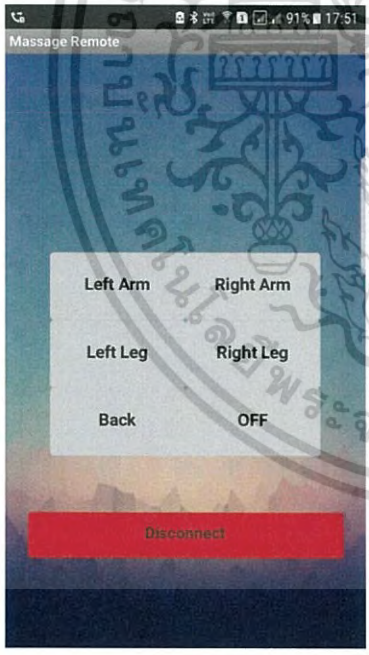
จากที่ได้ออกแบบแอปพลิเคชั่นในโปรแกรม App Inventor ตามบทที่ 3 ข้อที่ 3.2.4 โดยทดสอบการเชื่อมต่อกับอุปกรณ์บูลทูธ ส่งค่าตัวอักษรจากโทรศัพท์ไปยังอุปกรณ์บูลทูธ ผลลัพธ์ที่ได้คือ สามารถรับ-ส่ง ข้อมูลระหว่างกันได้โดยไม่มีปัญหาในการเชื่อมต่อ



รูปที่ 4.11 หน้าจอเริ่มต้น ไดโอด



รูปที่ 4.12 หน้าจอเลือกอุปกรณ์บูลทูธ



รูปที่ 4.13 หลังจากเลือกอุปกรณ์บูลทูธ

รูปที่ 4.14 หน้าจอเลือกระดับการสั่น

4.7) ผลการออกแบบและตัดเย็บชุดนวด

จากที่ได้ออกแบบชุดนวดในบทที่ 3 ข้อที่ 3.3) เพื่อนำมาตัดเย็บตามรูปแบบที่เขียนไว้ได้ผลดังรูป

รูปที่ 4.15 รูปชุดนวดหลังตัดเย็บและใส่หุ่นด้านหน้า

รูปที่ 4.16 รูปชุดนวดหลังตัดเย็บและใส่หุ่นด้านหลัง

4.8) ผลการทดสอบจังหวะในการสั่นสะเทือนที่ได้โปรแกรมไว้

จากที่ได้กล่าวไปในบทที่ 3 ข้อที่ 3.2.3) ได้ผลดังตารางต่อไปนี้

จังหวะที่	ON (วินาที)	OFF (วินาที)
1	1	1
2	1	1
3	1	1
4	1	1
5	4	1

ตารางที่ 4.6 เปรียบเทียบการใช้แรงดัน กระแส และกำลังไฟฟ้าของอุปกรณ์

โดยการทำงานจะเป็นดังตารางที่ 4.6 เมื่อจบการทำงานก็จะวนไปเริ่มจังหวะที่ 1 ใหม่ จนครบกำหนดเวลาที่ตั้งไว้

4.9) ผลการทดสอบการใช้กำลังไฟฟ้าทั้งหมด

โดยการทดสอบการวัดกระแสไฟฟ้าทั้งหมดที่ใช้ไปในระหว่างอุปกรณ์กำลังทำงานในฟังก์ชั่นต่างๆ ซึ่งได้ผลดังตารางต่อไปนี้

ส่วนที่ทำงาน	แรงดัน [V]	กระแส [A]	กำลังไฟฟ้า [W]
เริ่มต้น	12	6.4 mA	0.0768
1 ส่วน	12	0.143	1.716
2 ส่วน	12	0.169	2.028
3 ส่วน	12	0.245	2.94
4 ส่วน	12	0.317	3.804
5 ส่วน	12	0.427	5.124

ตารางที่ 4.7 เปรียบเทียบการใช้แรงดัน กระแส และกำลังไฟฟ้าของอุปกรณ์

บทที่ 5
สรุปผลการวิจัย และข้อเสนอแนะ

5.1) สรุปผลการทดลอง

จากขั้นตอนและผลการทดลองที่ได้กล่าวไปในบทที่ 3 และ บทที่ 4 แล้ว จะเห็นได้ว่าสามารถสร้างสัญญาณพัลส์ที่มีขนาดและความกว้างต่างๆได้ โดยการเขียนโปรแกรมภาษาซีและเบิร์นโปรแกรมลงในไมโครคอนโทรลเลอร์เบอร์ 16F887 ได้ โดยโปรแกรมภาษาซีที่เขียนนั้นจะใช้ได้เฉพาะไมโครคอนโทรลเลอร์ที่ได้ตั้งค่าไว้ในโปรแกรมเท่านั้น เมื่อได้สัญญาณพัลส์แล้วนำไปเชื่อมต่อกับมอเตอร์เพื่อทำงาน โดยไมโครคอนโทรลเลอร์นั้นจะอยู่ในวงจรควบคุมการทำงาน ซึ่งจะรับค่าสัญญาณอนาลอกที่มาจากสวิทช์ที่มีค่าความต่างศักย์ไม่เท่ากัน เมื่อไมโครคอนโทรลเลอร์ได้รับค่าสัญญาณอนาลอกแล้ว จะทำการประมวลผลค่าสัญญาณออกเป็นข้อมูล 10 บิท จากนั้นจะแปลงค่าที่ได้เป็นข้อมูลฐาน 10 หรือ จากข้อมูลที่ส่งผ่านสัญญาณบูลทูธผ่านโมดูล HC-06 ซึ่งสัญญาณข้อมูลจะถูกส่งมาจากโทรศัพท์มือถือระบบปฏิบัติการแอนดรอยด์ โดยจะส่งข้อมูลเป็นตัวหนังสือ เพื่อนำไปเปรียบเทียบกับเงื่อนไขที่ได้ตั้งไว้ เพื่อเปรียบเทียบเสร็จสิ้น ไมโครคอนโทรลเลอร์ก็จะส่งเอาท์พุทเป็นสัญญาณ PWM ซึ่งมี Duty Cycle ที่ 30% และ 60% ตามที่ตั้งไว้ในโปรแกรมภาษาซี ทำให้ค่าแรงดันมีค่าที่ต่างกันออกไปยังทรานซิสเตอร์ซึ่งมีหน้าที่เป็นวงจรขับกระแสของมอเตอร์ไฟฟ้ากระแสตรงเพื่อควบคุมความเร็วในการหมุน อีกทั้งยังส่งแรงสั่นสะเทือนบริเวณส่วนต่าง ๆ ของร่างกาย

5.2) ปัญหาที่พบ

1) ฟังก์ชั่นการใช้งานของแอพพลิเคชั่นยังไม่หลากหลาย
2) การตัดเย็บชุดนวดต้องใช้จักรเย็บหนังโดยเฉพาะ ไม่สามารถใช้จักรเย็บผ้าธรรมดาได้
3) โปรแกรมภาษาซีมีความซับซ้อนในการเข้าใจถึงการตั้งค่าต่างๆ
4) ฟังก์ชั่นการทำงานของชุดนวดยังไม่หลากหลาย

5.3) ข้อเสนอแนะ

1) พัฒนาแอพพลิเคชั่นเพิ่มเติมเพื่อให้มีความหลากหลาย
2) ศึกษาโปรแกรมภาษาซีจากโครงงานต่างๆ เพื่อนำไปต่อยอด
3) เพิ่มฟังก์ชั่นการทำงานขึ้นโดยอาจมีการใช้ระบบถุงลมเพื่อให้เหมือนการบีบนวดมากขึ้น

เอกสารอ้างอิง

[1] แรงสั่นสะเทือน (ออนไลน์). (2560). สืบค้นจาก : https://goo.gl/45UDgR [4 กุมภาพันธ์ 2561]

[2] ประโยชน์ของการสั่นสะเทือนทั่วร่างกาย (ออนไลน์). (2560).

สืบค้นจาก : https://goo.gl/fxnomm [4 กุมภาพันธ์ 2561]

[3] ความรู้เกี่ยวกับไมโครคอนโทรลเลอร์เบื้องต้น (ออนไลน์). (2560).

สืบค้นจาก : https://goo.gl/rPVo2s [4 กุมภาพันธ์ 2561]

[4] ไมโครคอนโทรลเลอร์เบื้องต้น (ออนไลน์). (2560).

สืบค้นจาก : https://goo.gl/ScjcF8 [4 กุมภาพันธ์ 2561]

[5] ความรู้เกี่ยวกับมอเตอร์ไฟฟ้า (ออนไลน์). (2560).

สืบค้นจาก : https:// http://202.129.59.73/tn/motor10-52/motor1.htm [4 กุมภาพันธ์ 2561]

[6] มอเตอร์ไฟฟ้ากระแสตรง (ออนไลน์). (2560).

สืบค้นจาก : http://montri.rmutl.ac.th/assets/dc06.pdf [4 กุมภาพันธ์ 2561]

[7] บลูทูธ "ฟันสีฟ้า" เทคโนโลยีไร้สายสำหรับอนาคต (ออนไลน์). (2560).

สืบค้นจาก : http://www.siamphone.com/news/bluetooth/page.htm [4 กุมภาพันธ์ 2561]

[8] ไอซีเร็กกูเลเตอร์ (IC REGULATOR) (ออนไลน์). (2560).

สืบค้นจาก : https://goo.gl/6hAA12 [4 กุมภาพันธ์ 2561]

[9] เริ่มใช้งาน MPLAB X IDE กับ HITECH-PICC (ออนไลน์). (2554).

สืบค้นจาก : https://goo.gl/p7cbKh [4 กุมภาพันธ์ 2561]

ภาคผนวก

ภาคผนวก ก.

IC Regulator LM7805

FAIRCHILD
SEMICONDUCTOR®

March 2008

LM78XX/LM78XXA
3-Terminal 1A Positive Voltage Regulator

Features

- Output Current up to 1A
- Output Voltages of 5, 6, 8, 9, 10, 12, 15, 18, 24
- Thermal Overload Protection
- Short Circuit Protection
- Output Transistor Safe Operating Area Protection

General Description

The LM78XX series of three terminal positive regulators are available in the TO-220 package and with several fixed output voltages, making them useful in a wide range of applications. Each type employs internal current limiting, thermal shut down and safe operating area protection, making it essentially indestructible. If adequate heat sinking is provided, they can deliver over 1A output current. Although designed primarily as fixed voltage regulators, these devices can be used with external components to obtain adjustable voltages and currents.

Ordering Information

Product Number	Output Voltage Tolerance	Package	Operating Temperature
LM7805CT	±4%	TO-220	-40°C to +125°C
LM7806CT			
LM7808CT			
LM7809CT			
LM7810CT			
LM7812CT			
LM7815CT			
LM7818CT			
LM7824CT			
LM7805ACT	±2%		0°C to +125°C
LM7806ACT			
LM7808ACT			
LM7809ACT			
LM7810ACT			
LM7812ACT			
LM7815ACT			
LM7818ACT			
LM7824ACT			

Block Diagram

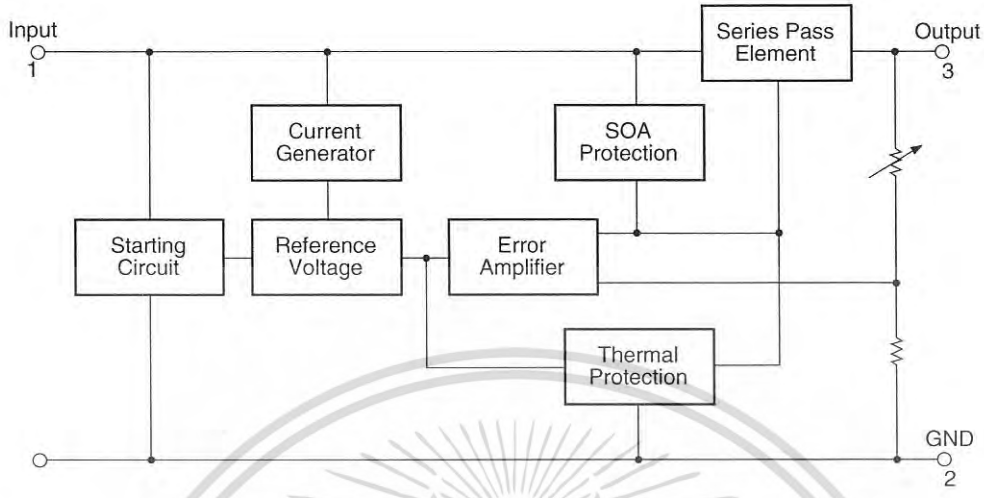


Figure 1.

Pin Assignment

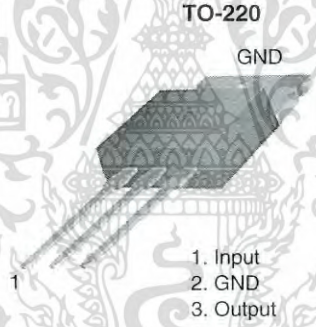


Figure 2.

Absolute Maximum Ratings

Absolute maximum ratings are those values beyond which damage to the device may occur. The datasheet specifications should be met, without exception, to ensure that the system design is reliable over its power supply, temperature, and output/input loading variables. Fairchild does not recommend operation outside datasheet specifications.

Symbol	Parameter		Value	Unit
V_I	Input Voltage	V_O = 5V to 18V	35	V
		V_O = 24V	40	V
$R_{\theta JC}$	Thermal Resistance Junction-Cases (TO-220)		5	°C/W
$R_{\theta JA}$	Thermal Resistance Junction-Air (TO-220)		65	°C/W
T_{OPR}	Operating Temperature Range	LM78xx	-40 to +125	°C
		LM78xxA	0 to +125	
T_{STG}	Storage Temperature Range		-65 to +150	°C

Electrical Characteristics (LM7805)

Refer to the test circuits. -40°C < T_J < 125°C, I_O = 500mA, V_I = 10V, C_I = 0.1µF, unless otherwise specified.

Symbol	Parameter	Conditions		Min.	Typ.	Max.	Unit
V_O	Output Voltage	T_J = +25°C		4.8	5.0	5.2	V
		5mA ≤ I_O ≤ 1A, P_O ≤ 15W, V_I = 7V to 20V		4.75	5.0	5.25	
Regline	Line Regulation(1)	T_J = +25°C	V_O = 7V to 25V	–	4.0	100	mV
			V_I = 8V to 12V	–	1.6	50.0	
Regload	Load Regulation(1)	T_J = +25°C	I_O = 5mA to 1.5A	–	9.0	100	mV
			I_O = 250mA to 750mA	–	4.0	50.0	
I_Q	Quiescent Current	T_J = +25°C		–	5.0	8.0	mA
ΔI_Q	Quiescent Current Change	I_O = 5mA to 1A		–	0.03	0.5	mA
		V_I = 7V to 25V		–	0.3	1.3	
$\Delta V_O/\Delta T$	Output Voltage Drift(2)	I_O = 5mA		–	-0.8	–	mV/°C
V_N	Output Noise Voltage	f = 10Hz to 100kHz, T_A = +25°C		–	42.0	–	µV/V_O
RR	Ripple Rejection(2)	f = 120Hz, V_O = 8V to 18V		62.0	73.0	–	dB
V_{DROP}	Dropout Voltage	I_O = 1A, T_J = +25°C		–	2.0	–	V
r_O	Output Resistance(2)	f = 1kHz		–	15.0	–	mΩ
I_{SC}	Short Circuit Current	V_I = 35V, T_A = +25°C		–	230	–	mA
I_{PK}	Peak Current(2)	T_J = +25°C		–	2.2	–	A

Notes:

1. Load and line regulation are specified at constant junction temperature. Changes in V_O due to heating effects must be taken into account separately. Pulse testing with low duty is used.
2. These parameters, although guaranteed, are not 100% tested in production.

Typical Performance Characteristics

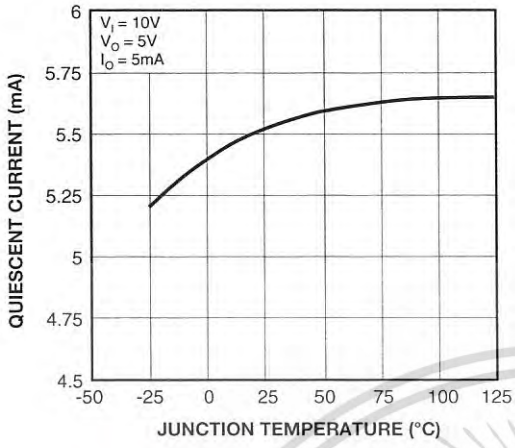


Figure 3. Quiescent Current

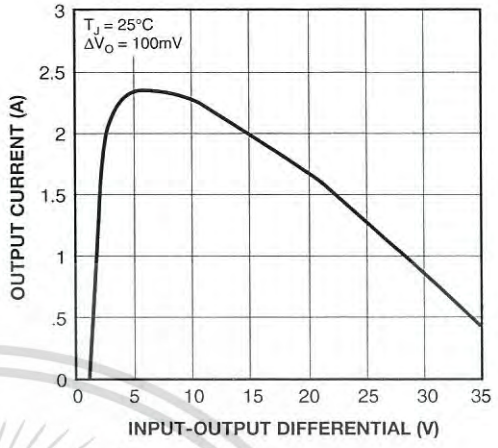


Figure 4. Peak Output Current

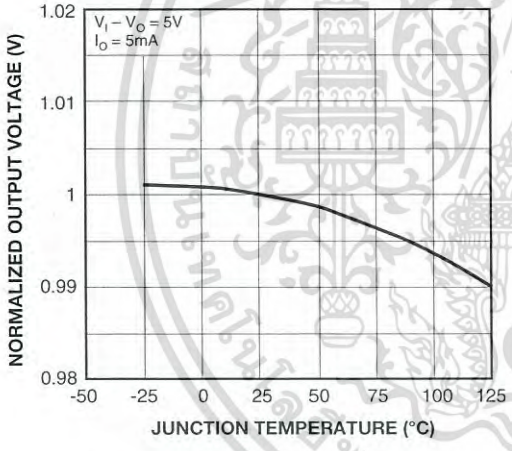


Figure 5. Output Voltage

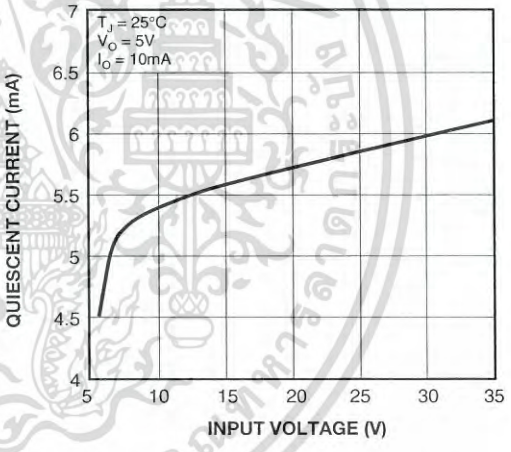


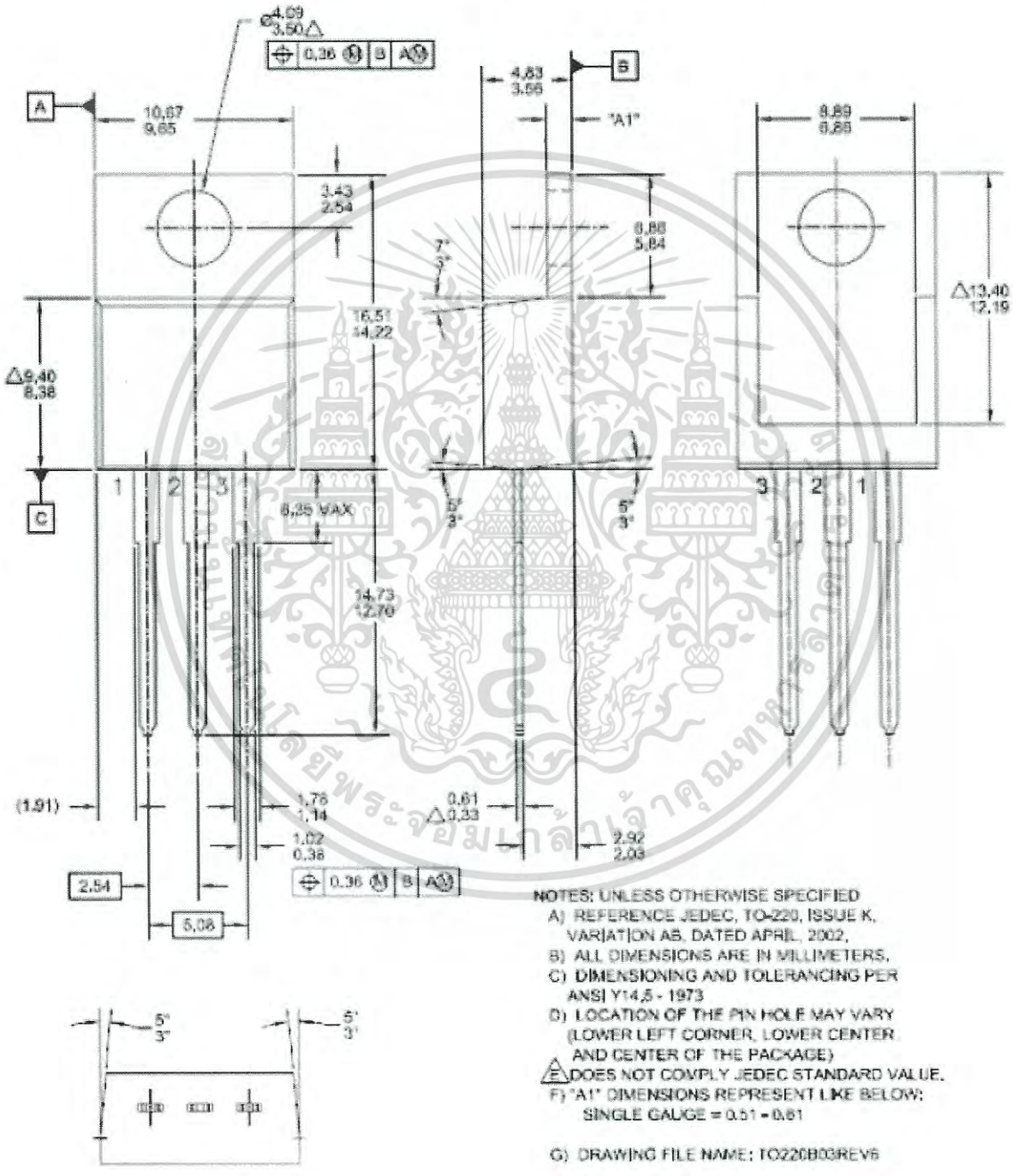
Figure 6. Quiescent Current

Mechanical Dimensions

Dimensions in millimeters

TO-220

Dimensions are in mm



NOTES: UNLESS OTHERWISE SPECIFIED

A) REFERENCE JEDEC, TO-220, ISSUE K, VARIATION AB, DATED APRIL, 2002.

B) ALL DIMENSIONS ARE IN MILLIMETERS.

C) DIMENSIONING AND TOLERANCING PER ANSI Y14.5 - 1973

D) LOCATION OF THE PIN HOLE MAY VARY (LOWER LEFT CORNER, LOWER CENTER AND CENTER OF THE PACKAGE)

△E DOES NOT COMPLY JEDEC STANDARD VALUE.

F) "A1" DIMENSIONS REPRESENT LIKE BELOW:
SINGLE GAUGE = 0.51 ~ 0.61

G) DRAWING FILE NAME: TO220B03REV6

ภาคผนวก ข.

Transistor KSD471A

FAIRCHILD SEMICONDUCTOR®

March 2013

KSD471A
NPN Epitaxial Silicon Transistor

Features

- Audio Frequency Power Amplifier
- Complement to KSB564A
- Collector Current: I_C = 1 A
- Collector Power Dissipation: P_C = 800 mW
- Suffix "-C" means Center Collector (1. Emitter 2. Collector 3. Base)

TO-92

1. Emitter 2. Base 3. Collector

Absolute Maximum Ratings

Stresses exceeding the absolute maximum ratings may damage the device. The device may not function or be operable above the recommended operating conditions and stressing the parts to these levels is not recommended. In addition, extended exposure to stresses above the recommended operating conditions may affect device reliability. The absolute maximum ratings are stress ratings only. Values are at T_A = 25°C unless otherwise noted.

Symbol	Parameter	Ratings	Unit
V_{CBO}	Collector-Base Voltage	40	V
V_{CEO}	Collector-Emitter Voltage	30	V
V_{EBO}	Emitter-Base Voltage	5	V
I_C	Collector Current	1	A
P_C	Collector Power Dissipation	800	mW
T_J	Junction Temperature	150	°C
T_{STG}	Storage Temperature	-55 to +150	°C

Electrical Characteristics

Values are at T_A = 25°C unless otherwise noted.

Symbol	Parameter	Test Condition	Min.	Typ.	Max.	Unit
BV_{CBO}	Collector-Base Breakdown Voltage	I_C = 100 μA, I_E = 0	40			V
BV_{CEO}	Collector-Emitter Breakdown Voltage	I_C = 10 mA, I_B = 0	30			V
BV_{EBO}	Emitter-Base Breakdown Voltage	I_E = 100 μA, I_C = 0	5			V
I_{CBO}	Collector Cut-off Current	V_{CB} = 30 V, I_E = 0			0.1	μA
h_{FE}	DC Current Gain	V_{CE} = 1 V, I_C = 100 mA	120		400	
V_{CE} (sat)	Collector-Emitter Saturation Voltage	I_C = 1 A, I_B = 0.1 A			0.5	V
V_{BE} (sat)	Base-Emitter Saturation Voltage	I_C = 1 A, I_B = 0.1 A			1.2	V
f_T	Current Gain BandWidth Product	V_{CE} = 6 V, I_C = 10 mA		130		MHz
C_{ob}	Output Capacitance	V_{CB} = 6 V, I_E = 0, f = 1 MHz		16		pF

h_{FE} Classification

Classification	Y	G
h_{FE}	120 ~ 240	200 ~ 400

Typical Performance Characteristics

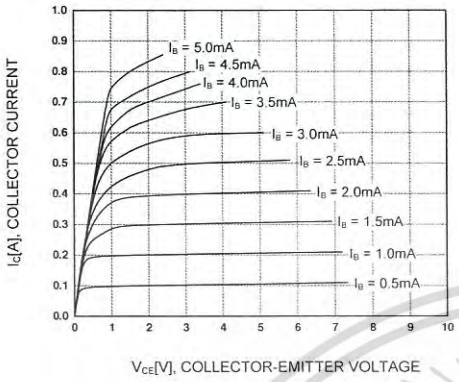


Figure 1. Static Characteristic

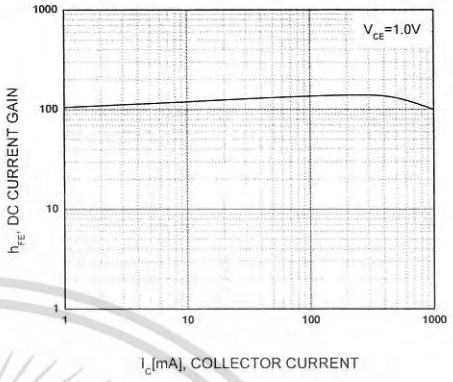


Figure 2. DC current Gain

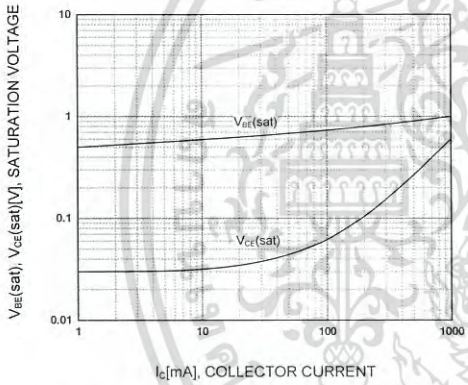


Figure 3. Base-Emitter Saturation Voltage
Collector-Emitter Saturation Voltage

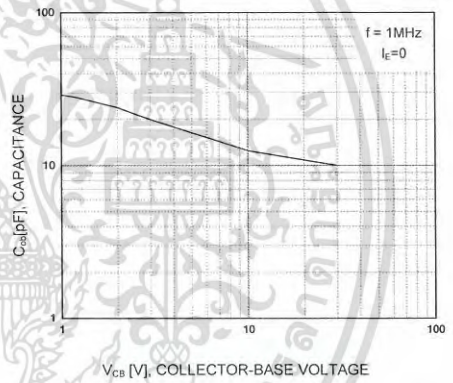


Figure 4. Collector Output Capacitance

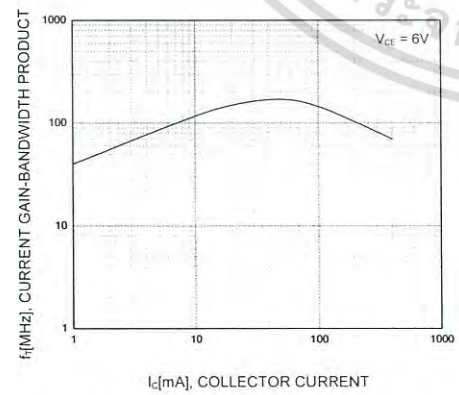


Figure 5. Current Gain Bandwidth Product

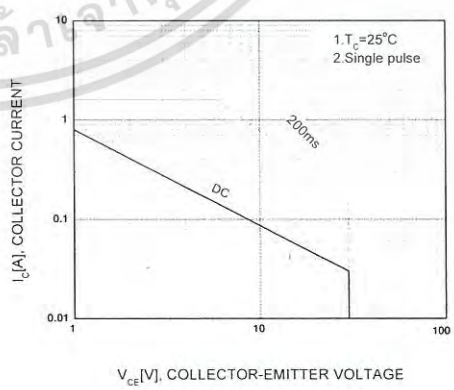


Figure 6. Safe Operating Area

Physical Dimensions

TO-92

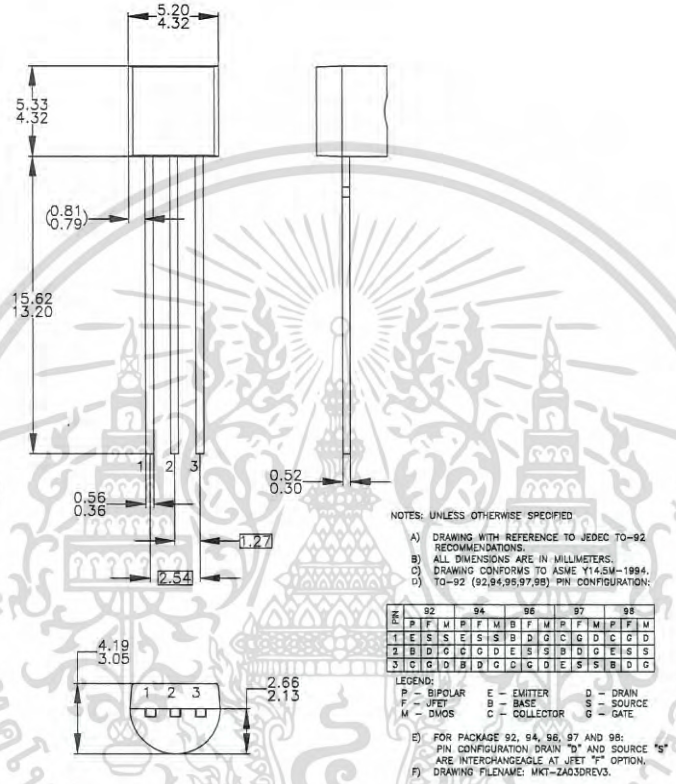


Figure 7. 3LEAD, TO92, JEDEC TO-92 COMPLIANT STRAIGHT LEAD CONFIGURATION (OLD TO92AM3)

Package drawings are provided as a service to customers considering Fairchild components. Drawings may change in any manner without notice. Please note the revision and/or date on the drawing and contact a Fairchild Semiconductor representative to verify or obtain the most recent revision. Package specifications do not expand the terms of Fairchild's worldwide terms and conditions, specifically the warranty therein, which covers Fairchild products.

Always visit Fairchild Semiconductor's online packaging area for the most recent package drawings:
http://www.fairchildsemi.com/packaging/.

ภาคผนวก ค.

Microncontroller PIC16F887

PIC16F882/883/884/886/887
Data Sheet

28/40/44-Pin, Enhanced Flash-Based 8-Bit CMOS Microcontrollers with nanoWatt Technology

PIC16F882/883/884/886/887

28/40/44-Pin Flash-Based, 8-Bit CMOS Microcontrollers with nanoWatt Technology

High-Performance RISC CPU:

- Only 35 Instructions to Learn:
  - All single-cycle instructions except branches
- Operating Speed:
  - DC – 20 MHz oscillator/clock input
  - DC – 200 ns instruction cycle
- Interrupt Capability
- 8-Level Deep Hardware Stack
- Direct, Indirect and Relative Addressing modes

Special Microcontroller Features:

- Precision Internal Oscillator:
  - Factory calibrated to ±1%
  - Software selectable frequency range of 8 MHz to 31 kHz
  - Software tunable
  - Two-Speed Start-up mode
  - Crystal fail detect for critical applications
  - Clock mode switching during operation for power savings
- Power-Saving Sleep mode
- Wide Operating Voltage Range (2.0V-5.5V)
- Industrial and Extended Temperature Range
- Power-on Reset (POR)
- Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Brown-out Reset (BOR) with Software Control Option
- Enhanced Low-Current Watchdog Timer (WDT) with On-Chip Oscillator (software selectable nominal 268 seconds with full prescaler) with software enable
- Multiplexed Master Clear with Pull-up/Input Pin
- Programmable Code Protection
- High Endurance Flash/EEPROM Cell:
  - 100,000 write Flash endurance
  - 1,000,000 write EEPROM endurance
  - Flash/Data EEPROM retention: > 40 years
- Program Memory Read/Write during run time
- In-Circuit Debugger (on board)

Low-Power Features:

- Standby Current:
  - 50 nA @ 2.0V, typical
- Operating Current:
  - 11 μA @ 32 kHz, 2.0V, typical
  - 220 μA @ 4 MHz, 2.0V, typical
- Watchdog Timer Current:
  - 1 μA @ 2.0V, typical

Peripheral Features:

- 24/35 I/O Pins with Individual Direction Control:
  - High current source/sink for direct LED drive
  - Interrupt-on-Change pin
  - Individually programmable weak pull-ups
  - Ultra Low-Power Wake-up (ULPWU)
- Analog Comparator Module with:
  - Two analog comparators
  - Programmable on-chip voltage reference (CV_{REF}) module (% of V_{DD})
  - Fixed voltage reference (0.6V)
  - Comparator inputs and outputs externally accessible
  - SR Latch mode
  - External Timer1 Gate (count enable)
- A/D Converter:
  - 10-bit resolution and 11/14 channels
- Timer0: 8-bit Timer/Counter with 8-bit Programmable Prescaler
- Enhanced Timer1:
  - 16-bit timer/counter with prescaler
  - External Gate Input mode
  - Dedicated low-power 32 kHz oscillator
- Timer2: 8-bit Timer/Counter with 8-bit Period Register, Prescaler and Postscaler
- Enhanced Capture, Compare, PWM+ Module:
  - 16-bit Capture, max. resolution 12.5 ns
  - Compare, max. resolution 200 ns
  - 10-bit PWM with 1, 2 or 4 output channels, programmable "dead time", max. frequency 20 kHz
  - PWM output steering control
- Capture, Compare, PWM Module:
  - 16-bit Capture, max. resolution 12.5 ns
  - 16-bit Compare, max. resolution 200 ns
  - 10-bit PWM, max. frequency 20 kHz
- Enhanced USART Module:
  - Supports RS-485, RS-232, and LIN 2.0
  - Auto-Baud Detect
  - Auto-Wake-Up on Start bit
- In-Circuit Serial Programming™ (ICSP™) via Two Pins
- Master Synchronous Serial Port (MSSP) Module supporting 3-wire SPI (all 4 modes) and I^2C™ Master and Slave Modes with I^2C Address Mask

Device	Program Memory	Data Memory		I/O	10-bit A/D (ch)	ECCP/ CCP	EUSART	MSSP	Comparators	Timers 8/16-bit
	Flash (words)	SRAM (bytes)	EEPROM (bytes)							
PIC16F882	2048	128	128	24	11	1/1	1	1	2	2/1
PIC16F883	4096	256	256	24	11	1/1	1	1	2	2/1
PIC16F884	4096	256	256	35	14	1/1	1	1	2	2/1
PIC16F886	8192	368	256	24	11	1/1	1	1	2	2/1
PIC16F887	8192	368	256	35	14	1/1	1	1	2	2/1

Pin Diagrams – PIC16F884/887, 40-Pin PDIP

40-pin PDIP

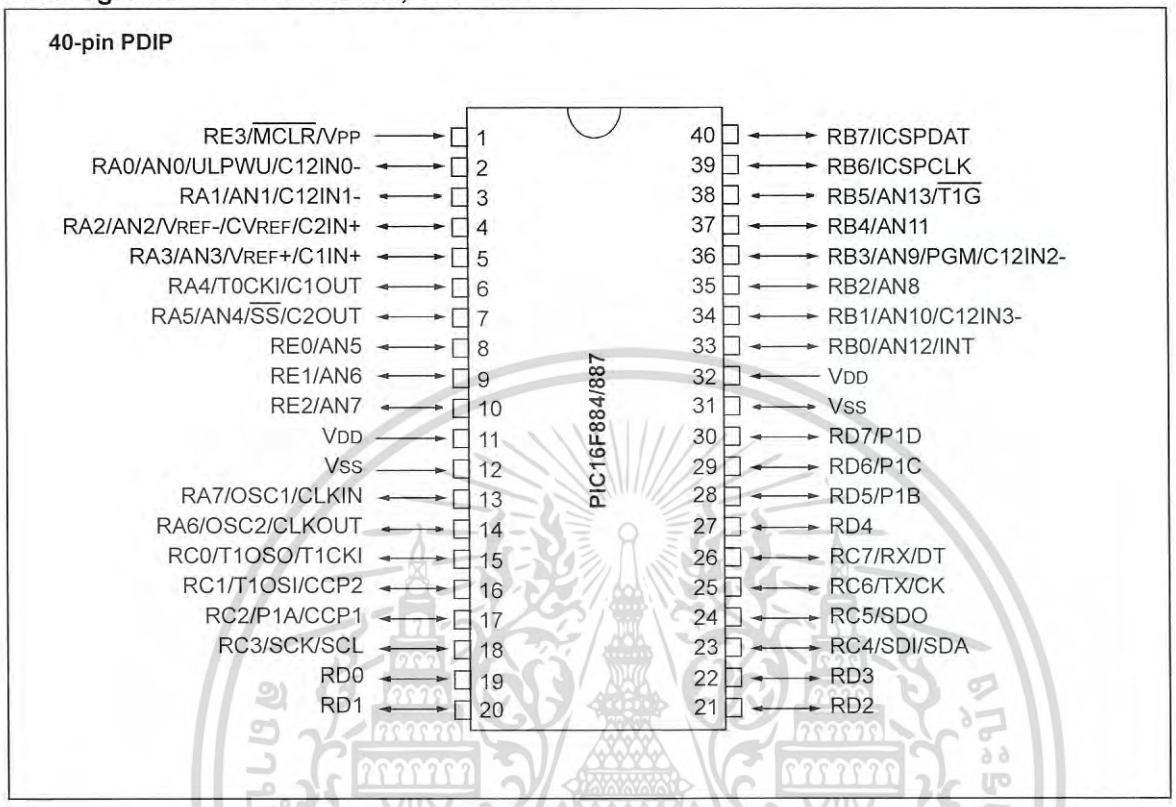
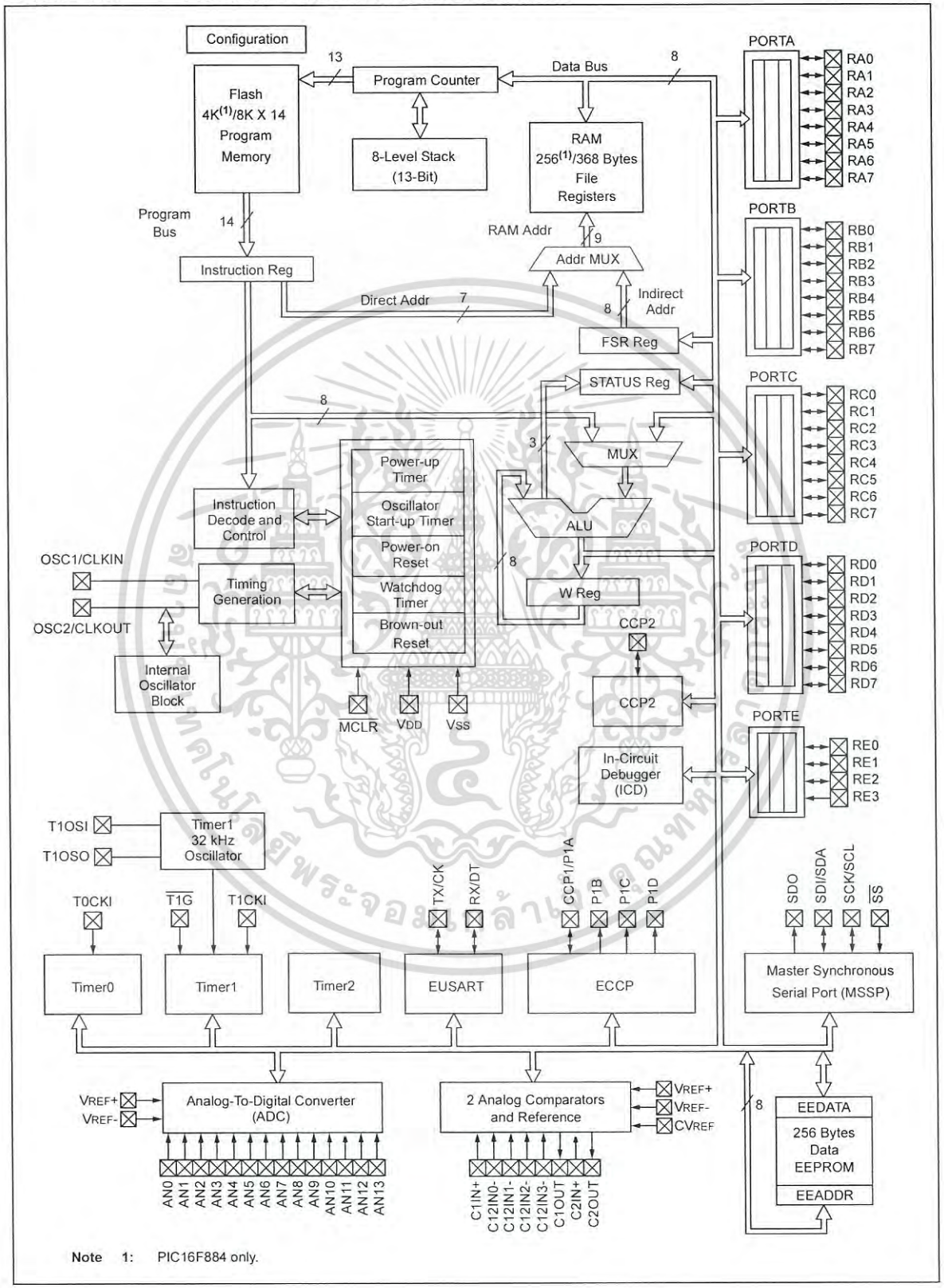


TABLE 3: PIC16F884/887 40-PIN SUMMARY (PDIP)

I/O	Pin	Analog	Comparators	Timers	ECCP	EUSART	MSSP	Interrupt	Pull-up	Basic
RA0	2	AN0/ULPWU	C12IN0-	—	—	—	—	—	—	—
RA1	3	AN1	C12IN1-	—	—	—	—	—	—	—
RA2	4	AN2	C2IN+	—	—	—	—	—	—	VREF-/CVREF
RA3	5	AN3	C1IN+	—	—	—	—	—	—	VREF+
RA4	6	—	C1OUT	T0CKI	—	—	—	—	—	—
RA5	7	AN4	C2OUT	—	—	—	$\overline{SS}$	—	—	—
RA6	14	—	—	—	—	—	—	—	—	OSC2/CLKOUT
RA7	13	—	—	—	—	—	—	—	—	OSC1/CLKIN
RB0	33	AN12	—	—	—	—	—	IOC/INT	Y	—
RB1	34	AN10	C12IN3-	—	—	—	—	IOC	Y	—
RB2	35	AN8	—	—	—	—	—	IOC	Y	—
RB3	36	AN9	C12IN2-	—	—	—	—	IOC	Y	PGM
RB4	37	AN11	—	—	—	—	—	IOC	Y	—
RB5	38	AN13	—	$\overline{T1G}$	—	—	—	IOC	Y	—
RB6	39	—	—	—	—	—	—	IOC	Y	ICSPCLK
RB7	40	—	—	—	—	—	—	IOC	Y	ICSPDAT
RC0	15	—	—	T1OSO/T1CKI	—	—	—	—	—	—
RC1	16	—	—	T1OSI	CCP2	—	—	—	—	—
RC2	17	—	—	—	CCP1/P1A	—	—	—	—	—
RC3	18	—	—	—	—	—	SCK/SCL	—	—	—
RC4	23	—	—	—	—	—	SDI/SDA	—	—	—
RC5	24	—	—	—	—	—	SDO	—	—	—
RC6	25	—	—	—	—	TX/CK	—	—	—	—
RC7	26	—	—	—	—	RX/DT	—	—	—	—
RD0	19	—	—	—	—	—	—	—	—	—
RD1	20	—	—	—	—	—	—	—	—	—
RD2	21	—	—	—	—	—	—	—	—	—
RD3	22	—	—	—	—	—	—	—	—	—
RD4	27	—	—	—	—	—	—	—	—	—
RD5	28	—	—	—	P1B	—	—	—	—	—
RD6	29	—	—	—	P1C	—	—	—	—	—
RD7	30	—	—	—	P1D	—	—	—	—	—
RE0	8	AN5	—	—	—	—	—	—	—	—
RE1	9	AN6	—	—	—	—	—	—	—	—
RE2	10	AN7	—	—	—	—	—	—	—	—
RE3	1	—	—	—	—	—	—	—	Y(1)	$\overline{MCLR}$/VPP
—	11	—	—	—	—	—	—	—	—	VDD
—	32	—	—	—	—	—	—	—	—	VDD
—	12	—	—	—	—	—	—	—	—	VSS
—	31	—	—	—	—	—	—	—	—	VSS

Note 1: Pull-up activated only with external $\overline{MCLR}$ configuration.

FIGURE 1-2: PIC16F884/PIC16F887 BLOCK DIAGRAM



Note 1: PIC16F884 only.

TABLE 1-2: PIC16F884/887 PINOUT DESCRIPTION

Name	Function	Input Type	Output Type	Description
RA0/AN0/ULPWU/C12IN0-	RA0	TTL	CMOS	General purpose I/O.
	AN0	AN	—	A/D Channel 0 input.
	ULPWU	AN	—	Ultra Low-Power Wake-up input.
	C12IN0-	AN	—	Comparator C1 or C2 negative input.
RA1/AN1/C12IN1-	RA1	TTL	CMOS	General purpose I/O.
	AN1	AN	—	A/D Channel 1 input.
	C12IN1-	AN	—	Comparator C1 or C2 negative input.
RA2/AN2/VREF-/CVREF/C2IN+	RA2	TTL	CMOS	General purpose I/O.
	AN2	AN	—	A/D Channel 2.
	VREF-	AN	—	A/D Negative Voltage Reference input.
	CVREF	—	AN	Comparator Voltage Reference output.
	C2IN+	AN	—	Comparator C2 positive input.
RA3/AN3/VREF+/C1IN+	RA3	TTL	CMOS	General purpose I/O.
	AN3	AN	—	A/D Channel 3.
	VREF+	AN	—	A/D Positive Voltage Reference input.
	C1IN+	AN	—	Comparator C1 positive input.
RA4/T0CKI/C1OUT	RA4	TTL	CMOS	General purpose I/O.
	T0CKI	ST	—	Timer0 clock input.
	C1OUT	—	CMOS	Comparator C1 output.
RA5/AN4/$\overline{SS}$/C2OUT	RA5	TTL	CMOS	General purpose I/O.
	AN4	AN	—	A/D Channel 4.
	$\overline{SS}$	ST	—	Slave Select input.
	C2OUT	—	CMOS	Comparator C2 output.
RA6/OSC2/CLKOUT	RA6	TTL	CMOS	General purpose I/O.
	OSC2	—	XTAL	Crystal/Resonator.
	CLKOUT	—	CMOS	FOSC/4 output.
RA7/OSC1/CLKIN	RA7	TTL	CMOS	General purpose I/O.
	OSC1	XTAL	—	Crystal/Resonator.
	CLKIN	ST	—	External clock input/RC oscillator connection.
RB0/AN12/INT	RB0	TTL	CMOS	General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
	AN12	AN	—	A/D Channel 12.
	INT	ST	—	External interrupt.
RB1/AN10/C12IN3-	RB1	TTL	CMOS	General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
	AN10	AN	—	A/D Channel 10.
	C12IN3-	AN	—	Comparator C1 or C2 negative input.
RB2/AN8	RB2	TTL	CMOS	General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
	AN8	AN	—	A/D Channel 8.
RB3/AN9/PGM/C12IN2-	RB3	TTL	CMOS	General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
	AN9	AN	—	A/D Channel 9.
	PGM	ST	—	Low-voltage ICSP™ Programming enable pin.
	C12IN2-	AN	—	Comparator C1 or C2 negative input.

Legend: AN = Analog input or output CMOS = CMOS compatible input or output OD = Open Drain
TTL = TTL compatible input ST = Schmitt Trigger input with CMOS levels
HV = High Voltage XTAL = Crystal

TABLE 1-2: PIC16F884/887 PINOUT DESCRIPTION (CONTINUED)

Name	Function	Input Type	Output Type	Description
RB4/AN11	RB4	TTL	CMOS	General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
	AN11	AN	—	A/D Channel 11.
RB5/AN13/$\overline{T1G}$	RB5	TTL	CMOS	General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
	AN13	AN	—	A/D Channel 13.
	$\overline{T1G}$	ST	—	Timer1 Gate input.
RB6/ICSPCLK	RB6	TTL	CMOS	General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
	ICSPCLK	ST	—	Serial Programming Clock.
RB7/ICSPDAT	RB7	TTL	CMOS	General purpose I/O. Individually controlled interrupt-on-change. Individually enabled pull-up.
	ICSPDAT	ST	TTL	ICSP™ Data I/O.
RC0/T1OSO/T1CKI	RC0	ST	CMOS	General purpose I/O.
	T1OSO	—	XTAL	Timer1 oscillator output.
	T1CKI	ST	—	Timer1 clock input.
RC1/T1OSI/CCP2	RC1	ST	CMOS	General purpose I/O.
	T1OSI	XTAL	—	Timer1 oscillator input.
	CCP2	ST	CMOS	Capture/Compare/PWM2.
RC2/P1A/CCP1	RC2	ST	CMOS	General purpose I/O.
	P1A	ST	CMOS	PWM output.
	CCP1	—	CMOS	Capture/Compare/PWM1.
RC3/SCK/SCL	RC3	ST	CMOS	General purpose I/O.
	SCK	ST	CMOS	SPI clock.
	SCL	ST	OD	I^2C™ clock.
RC4/SDI/SDA	RC4	ST	CMOS	General purpose I/O.
	SDI	ST	—	SPI data input.
	SDA	ST	OD	I^2C data input/output.
RC5/SDO	RC5	ST	CMOS	General purpose I/O.
	SDO	—	CMOS	SPI data output.
RC6/TX/CK	RC6	ST	CMOS	General purpose I/O.
	TX	—	CMOS	EUSART asynchronous transmit.
	CK	ST	CMOS	EUSART synchronous clock.
RC7/RX/DT	RC7	ST	CMOS	General purpose I/O.
	RX	ST	—	EUSART asynchronous input.
	DT	ST	CMOS	EUSART synchronous data.
RD0	RD0	TTL	CMOS	General purpose I/O.
RD1	RD1	TTL	CMOS	General purpose I/O.
RD2	RD2	TTL	CMOS	General purpose I/O.
RD3	RD3	TTL	CMOS	General purpose I/O.
RD4	RD4	TTL	CMOS	General purpose I/O.
RD5/P1B	RD5	TTL	CMOS	General purpose I/O.
	P1B	—	CMOS	PWM output.
RD6/P1C	RD6	TTL	CMOS	General purpose I/O.
	P1C	—	CMOS	PWM output.

Legend: AN = Analog input or output; CMOS = CMOS compatible input or output; OD = Open Drain
TTL = TTL compatible input; ST = Schmitt Trigger input with CMOS levels
HV = High Voltage; XTAL = Crystal

TABLE 1-2: PIC16F884/887 PINOUT DESCRIPTION (CONTINUED)

Name	Function	Input Type	Output Type	Description
RD7/P1D	RD7	TTL	CMOS	General purpose I/O.
	P1D	AN	—	PWM output.
RE0/AN5	RE0	TTL	CMOS	General purpose I/O.
	AN5	AN	—	A/D Channel 5.
RE1/AN6	RE1	TTL	CMOS	General purpose I/O.
	AN6	AN	—	A/D Channel 6.
RE2/AN7	RE2	TTL	CMOS	General purpose I/O.
	AN7	AN	—	A/D Channel 7.
RE3/$\overline{MCLR}$/VPP	RE3	TTL	—	General purpose input.
	$\overline{MCLR}$	ST	—	Master Clear with internal pull-up.
	VPP	HV	—	Programming voltage.
VSS	VSS	Power	—	Ground reference.
VDD	VDD	Power	—	Positive supply.

Legend: AN = Analog input or output CMOS = CMOS compatible input or output OD = Open Drain
TTL = TTL compatible input ST = Schmitt Trigger input with CMOS levels
HV = High Voltage XTAL = Crystal

ภาคผนวก ง.

C Programming Code

C:\Users\Apracha\Dropbox\CODE\16F887 Massage Real\code.c

```
1
2   /* Main.c file generated by New Project wizard
3    *
4    * Created:   15/12/2017 Fri.
5    * Processor: PIC16F887
6    * Compiler:  HI-TECH C for PIC10/12/16
7    */
8   #include <pic.h>
9   __CONFIG(INTIO & WDTDIS & MCLREN & LVPDIS);
10  #define _XTAL_FREQ 4000000 // 4 MHz
11
12
13  #define TIME_DEBOUNCE 500
14  #define PWM_LEVEL_1 3
15  #define PWM_LEVEL_2 6
16  #define PWM_LEVEL_3 8
17  #define PWM_LEVEL_4 9
18  #define TIME_PERIOD_PWM 10
19
20  #define LIMIT_TIME_PROCESS_1          8
21  #define LIMIT_TIME_PROCESS_2          7
22  #define LIMIT_TIME_PROCESS_3          6
23  #define LIMIT_TIME_PROCESS_4          5
24
25  #define TIME_MOTOR_1_2_ON_SWING 1000
26  #define TIMES_MOTOR_SWING 10 // 4 times
27  #define TIMES_MOTOR_NOT_SWING 4
28  // TIMES_MOTOR_SWING + TIMES_MOTOR_NOT_SWING = total
29
30  #define OFFSET 20
31  #define REF_KEY_1 25 + OFFSET //0v
32  #define REF_KEY_2 365 + OFFSET //1.79v
33  #define REF_KEY_3 621 + OFFSET //3.04v
34  #define REF_KEY_4 771 + OFFSET //3.77v
35  #define REF_KEY_5 860 + OFFSET // 4.2v
36  #define REF_KEY_6 937 + OFFSET // 4.58v
37  //(0.44*1024)/5
38
39  #define LIMIT_TIMEOUT           15*60
40
41  #define MOTOR1_TRIS TRISD0
42  #define MOTOR2_TRIS TRISD1
43  #define MOTOR3_TRIS TRISD3
44  #define MOTOR4_TRIS TRISD2
45  #define MOTOR5_TRIS TRISC4
46  #define MOTOR6_TRIS TRISC5
47  #define MOTOR7_TRIS TRISD4
48  #define MOTOR8_TRIS TRISD5
49  #define MOTOR9_TRIS TRISD6
50  #define MOTOR10_TRIS TRISD7
51  #define MOTOR11_TRIS TRISB0
52  #define MOTOR12_TRIS TRISB1
53
```

```
54  #define MOTOR1 RD0
55  #define MOTOR2 RD1
56  #define MOTOR3 RD3
57  #define MOTOR4 RD2
58  #define MOTOR5 RC4
59  #define MOTOR6 RC5
60  #define MOTOR7 RD4
61  #define MOTOR8 RD5
62  #define MOTOR9 RD6
63  #define MOTOR10 RD7
64  #define MOTOR11 RB0
65  #define MOTOR12 RB1
66
67
68  int fn_read_key(void);
69  int fn_read_BT(void);
70  void Initial_timer1(void);
71  void Initial_timer2(void);
72  void Initial_Ports(void);
73  void Initial_ADC(void);
74  void All_OFF(void);
75  void motor_Arm1  (unsigned char level);
76  void motor_Arm2  (unsigned char level);
77  void motor_Leg1  (unsigned char level);
78  void motor_Leg2  (unsigned char level);
79  void motor_Body  (unsigned char level);
80  unsigned int analog(unsigned char channel);
81
82  void Initial_Bluetooth(void);
83  void BT_putc(unsigned char c);
84  void BT_puts(unsigned char *s);
85  unsigned char BT_getc();
86  void BT_Process(unsigned char process);
87
88
89  struct ccount{
90
91      unsigned int timeout;
92      unsigned int timeoutMsec;
93
94      unsigned int levelPWM_Arm1;
95      unsigned int levelPWM_Arm2;
96      unsigned int levelPWM_Leg1;
97      unsigned int levelPWM_Leg2;
98      unsigned int timesMotorOn;
99      unsigned int timerMotorOn;
100     unsigned int levelPWM_Body;
101
102     unsigned int readKey;
103
104     unsigned int process_2_MArm1;
105     unsigned int process_2_MArm2;
106     unsigned int process_2_MLeg1;
```

```
107         unsigned int process_2_MLeg2;
108         unsigned int process_2_MBody;
109
110         unsigned int process_3_MArm1;
111         unsigned int process_3_MArm2;
112         unsigned int process_3_MLeg1;
113         unsigned int process_3_MLeg2;
114         unsigned int process_3_MBody;
115
116         unsigned int Arm1_low;
117         unsigned int Arm2_low;
118         unsigned int Leg1_low;
119         unsigned int Leg2_low;
120         unsigned int Back_low;
121         unsigned int OFF;
122
123         unsigned int Arm1_high;
124         unsigned int Arm2_high;
125         unsigned int Leg1_high;
126         unsigned int Leg2_high;
127         unsigned int Back_high;
128
129     }   count;
130     struct
131     {
132         unsigned int timeout_Arm1;
133         unsigned int timeoutMsec_Arm1;
134         unsigned int timeout_Arm2;
135         unsigned int timeoutMsec_Arm2;
136         unsigned int timeout_Leg1;
137         unsigned int timeoutMsec_Leg1;
138         unsigned int timeout_Leg2;
139         unsigned int timeoutMsec_Leg2;
140         unsigned int timeout_Body;
141         unsigned int timeoutMsec_Body;
142         unsigned int timesMotorArm1On;
143         unsigned int timerMotorArm1On;
144         unsigned int timesMotorArm2On;
145         unsigned int timerMotorArm2On;
146         unsigned int timesMotorLeg1On;
147         unsigned int timerMotorLeg1On;
148         unsigned int timesMotorLeg2On;
149         unsigned int timerMotorLeg2On;
150         unsigned int timesMotorBodyOn;
151         unsigned int timerMotorBodyOn;
152     }   countMass;
153     struct status{
154         char M1;
155         char M2;
156         char M3;
157         char M4;
158         char M5;
159         char Button1;
```

```
160         char Button2;
161         char Button3;
162         char Button4;
163         char Button5;
164
165         char timeout;
166
167         char EnMotorArm1;
168         char EnMotorArm2;
169         char EnMotorLeg1;
170         char EnMotorLeg2;
171         char EnMotorBody;
172         char settimerMotorOn;
173         char settimerMotorArm1On;
174         char settimerMotorArm2On;
175         char settimerMotorLeg1On;
176         char settimerMotorLeg2On;
177         char settimerMotorBodyOn;
178     }   flag;
179     enum Statee
180     {
181         INITIAL_STATE,
182         RUNNING_STATE,
183         btnKEY1,
184         btnKEY2,
185         btnKEY3,
186         btnKEY4,
187         btnKEY5,
188         btnKEY6,
189         btnNONE,
190     };
191     char state;
192     unsigned int read_adc = 0;
193     unsigned char str[6],read_BT;
194     ////////////////////A
        Timer//////////////////////////////////////////////////////
        //////////
195     unsigned int count_timeMs = 0,count_timeSec = 0;
196     unsigned int count_timeMs2 = 0,count_timeSec2 = 0;
197     unsigned int count_timeMs_Body = 0,count_timeSec_Body = 0;
198     unsigned int count_timeMs2_Body = 0,count_timeSec2_Body = 0;
199     unsigned int count_timeMs_Arm1 = 0,count_timeSec_Arm1 = 0;
200     unsigned int count_timeMs2_Arm1 = 0,count_timeSec2_Arm1 = 0;
201     unsigned int count_timeMs_Arm2 = 0,count_timeSec_Arm2 = 0;
202     unsigned int count_timeMs2_Arm2 = 0,count_timeSec2_Arm2 = 0;
203     unsigned int count_timeMs_Leg1 = 0,count_timeSec_Leg1 = 0;
204     unsigned int count_timeMs2_Leg1 = 0,count_timeSec2_Leg1 = 0;
205     unsigned int count_timeMs_Leg2 = 0,count_timeSec_Leg2 = 0;
206     unsigned int count_timeMs2_Leg2 = 0,count_timeSec2_Leg2 = 0;
207
208     unsigned int count_timePeriodMotor_Body = 0;
209     unsigned int count_timePeriodMotor_Arm1 = 0;
210     unsigned int count_timePeriodMotor_Arm2 = 0;
```

```
211  unsigned int count_timePeriodMotor_Leg1 = 0;
212  unsigned int count_timePeriodMotor_Leg2 = 0;
213
214  //////////////////////A
     Timer//////////////////////////////////////////////////////////
     //////////
215
216  static void interrupt isr(void)
217  {
218      if(TMR1IF == 1) // interval every 1 ms // 125 count
219      {
220          TMR1IF = 0; //clear timer overflow flag
221          TMR1H = 0xFF;
222          TMR1L = 0x82; // 0xFF82 count every 1 ms
223
224          if(flag.M1 == 1)
225          {
226              count_timeMs_Arm1++;
227              if(count_timeMs_Arm1 >= 1000)
228              {
229                  count_timeMs_Arm1 = 0;
230                  if(count_timeSec_Arm1 > 0)
231                  {
232                      count_timeSec_Arm1--;
233                  }
234              }
235              count_timeMs2_Arm1++;
236              if(count_timeMs2_Arm1 >= 1000)
237              {
238                  count_timeMs2_Arm1 = 0;
239                  if(count_timeSec2_Arm1 > 0)
240                  {
241                      count_timeSec2_Arm1--;
242                  }
243              }
244              countMass.timeoutMsec_Arm1++;
245              if(countMass.timeoutMsec_Arm1 >= 1000)
246              {
247                  countMass.timeoutMsec_Arm1 = 0;
248                  if(countMass.timeout_Arm1 > 0)
249                  {
250                      countMass.timeout_Arm1--;
251                  }
252                  else if(countMass.timeout_Arm1 == 0)
253                  {
254                      flag.M1 = 0;
255                  }
256              }
257          }
258  ////////////////////////////////////////////////////////////////
     ////////////////////////////////
259      if(flag.M2 == 1)
260
```

```
261                 count_timeMs_Arm2++;
262                 if(count_timeMs_Arm2 >= 1000)
263                 {
264                     count_timeMs_Arm2 = 0;
265                     if(count_timeSec_Arm2 > 0)
266                     {
267                         count_timeSec_Arm2--;
268                     }
269                 }
270                 count_timeMs2_Arm2++;
271                 if(count_timeMs2_Arm2 >= 1000)
272                 {
273                     count_timeMs2_Arm2 = 0;
274                     if(count_timeSec2_Arm2 > 0)
275                     {
276                         count_timeSec2_Arm2--;
277                     }
278                 }
279                 countMass.timeoutMsec_Arm2++;
280                 if(countMass.timeoutMsec_Arm2 >= 1000)
281                 {
282                     countMass.timeoutMsec_Arm2 = 0;
283                     if(countMass.timeout_Arm2 > 0)
284                     {
285                         countMass.timeout_Arm2--;
286                     }
287                     else if(countMass.timeout_Arm2 == 0)
288                     {
289                         flag.M2 = 0;
290                     }
291                 }
292             }
293 ////////////////////////////////////////////////////////////////////////////
    ////////////////////////////////
294         if(flag.M3 == 1)
295         {
296             count_timeMs_Leg1++;
297             if(count_timeMs_Leg1 >= 1000)
298             {
299                 count_timeMs_Leg1 = 0;
300                 if(count_timeSec_Leg1 > 0)
301                 {
302                     count_timeSec_Leg1--;
303                 }
304             }
305             count_timeMs2_Leg1++;
306             if(count_timeMs2_Leg1 >= 1000)
307             {
308                 count_timeMs2_Leg1 = 0;
309                 if(count_timeSec2_Leg1 > 0)
310                 {
311                     count_timeSec2_Leg1--;
312
```

```
313                 }
314                 countMass.timeoutMsec_Leg1++;
315                 if(countMass.timeoutMsec_Leg1 >= 1000)
316                 {
317                     countMass.timeoutMsec_Leg1 = 0;
318                     if(countMass.timeout_Leg1 > 0)
319                         {
320                             countMass.timeout_Leg1--;
321                         }
322                     else if(countMass.timeout_Leg1 == 0)
323                     {
324                         flag.M3 = 0;
325                     }
326                 }
327             }
328     ////////////////////////////////////////////////////////////////////
        ////////////////////////////////
329             if(flag.M4 == 1)
330             {
331                 count_timeMs_Leg2++;
332                 if(count_timeMs_Leg2 >= 1000)
333                 {
334                     count_timeMs_Leg2 = 0;
335                     if(count_timeSec_Leg2 > 0)
336                     {
337                         count_timeSec_Leg2--;
338                     }
339                 }
340                 count_timeMs2_Leg2++;
341                 if(count_timeMs2_Leg2 >= 1000)
342                 {
343                     count_timeMs2_Leg2 = 0;
344                     if(count_timeSec2_Leg2 > 0)
345                     {
346                         count_timeSec2_Leg2--;
347                     }
348                 }
349                 countMass.timeoutMsec_Leg2++;
350                 if(countMass.timeoutMsec_Leg2 >= 1000)
351                 {
352                     countMass.timeoutMsec_Leg2 = 0;
353                     if(countMass.timeout_Leg2 > 0)
354                         {
355                             countMass.timeout_Leg2--;
356                         }
357                     else if(countMass.timeout_Leg2 == 0)
358                     {
359                         flag.M4 = 0;
360                     }
361                 }
362             }
363 ////////////////////////////////////////////////////////////////////
    ////////////////////////////////
```

```
364             if(flag.M5 == 1)
365             {
366                 count_timeMs_Body++;
367                 if(count_timeMs_Body >= 1000)
368                 {
369                     count_timeMs_Body = 0;
370                     if(count_timeSec_Body > 0)
371                     {
372                         count_timeSec_Body--;
373                     }
374                 }
375                 count_timeMs2_Body++;
376                 if(count_timeMs2_Body >= 1000)
377                 {
378                     count_timeMs2_Body = 0;
379                     if(count_timeSec2_Body > 0)
380                     {
381                         count_timeSec2_Body--;
382                     }
383                 }
384                 countMass.timeoutMsec_Body++;
385                 if(countMass.timeoutMsec_Body >= 1000)
386                 {
387                     countMass.timeoutMsec_Body = 0;
388                     if(countMass.timeout_Body > 0)
389                     {
390                         countMass.timeout_Body--;
391                     }
392                     else if(countMass.timeout_Body <= 0)
393                     {
394                         flag.M5 = 0;
395                     }
396 
397                 }
398 ////////////////////////////////////////////////////////////////////
    ////////////////////////////////
399             if(countMass.timerMotorArm1On == 0 )
400             {
401                 countMass.timerMotorArm1On =
    TIME_MOTOR_1_2_ON_SWING;
402                 countMass.timesMotorArm1On++;
403                 if(countMass.timesMotorArm1On <
    TIMES_MOTOR_SWING )
404                 {
405                     flag.settimerMotorArm1On ^= 1;
406                 }
407                 else if(countMass.timesMotorArm1On <
    TIMES_MOTOR_SWING+TIMES_MOTOR_NOT_SWING )
408                 {
409                     flag.settimerMotorArm1On = 1;
410                 }
411                 else
412 
```

```
413                 countMass.timesMotorArm1On = 0;
414             }
415         }
416         else if(countMass.timerMotorArm1On > 0 ){
    countMass.timerMotorArm1On --;}
417 ////////////////////////////////////////////////////////////////
    ////////////////////////////////
418     if(countMass.timerMotorArm2On == 0 )
419         {
420             countMass.timerMotorArm2On =
    TIME_MOTOR_1_2_ON_SWING;
421             countMass.timesMotorArm2On++;
422             if(countMass.timesMotorArm2On <
    TIMES_MOTOR_SWING )
423             {
424                 flag.settimerMotorArm2On ^= 1;
425             }
426             else if(countMass.timesMotorArm2On <
    TIMES_MOTOR_SWING+TIMES_MOTOR_NOT_SWING )
427             {
428                 flag.settimerMotorArm2On = 1;
429             }
430             else
431             {
432                 countMass.timesMotorArm2On = 0;
433             }
434
435         else if(countMass.timerMotorArm2On > 0 )
    countMass.timerMotorArm2On --;}
436 ////////////////////////////////////////////////////////////////
    ////////////////////////////////
437     if(countMass.timerMotorLeg1On == 0 )
438         {
439             countMass.timerMotorLeg1On =
    TIME_MOTOR_1_2_ON_SWING;
440             countMass.timesMotorLeg1On++;
441             if(countMass.timesMotorLeg1On <
    TIMES_MOTOR_SWING )
442             {
443                 flag.settimerMotorLeg1On ^= 1;
444             }
445             else if(countMass.timesMotorLeg1On <
    TIMES_MOTOR_SWING+TIMES_MOTOR_NOT_SWING )
446             {
447                 flag.settimerMotorLeg1On = 1;
448             }
449             else
450             {
451                 countMass.timesMotorLeg1On = 0;
452             }
453         }
454         else if(countMass.timerMotorLeg1On > 0 )
    countMass.timerMotorLeg1On --;}
```

```
455 ////////////////////////////////////////////////////////////////////////////////
456 if(countMass.timerMotorLeg2On == 0 )
457         {
458             countMass.timerMotorLeg2On =
    TIME_MOTOR_1_2_ON_SWING;
459             countMass.timesMotorLeg2On++;
460             if(countMass.timesMotorLeg2On <
    TIMES_MOTOR_SWING )
461             {
462                 flag.settimerMotorLeg2On ^= 1;
463             }
464             else if(countMass.timesMotorLeg2On <
    TIMES_MOTOR_SWING+TIMES_MOTOR_NOT_SWING )
465             {
466                 flag.settimerMotorLeg2On = 1;
467             }
468             else
469             {
470                 countMass.timesMotorLeg2On = 0;
471             }
472         }
473         else if(countMass.timerMotorLeg2On > 0 ){
    countMass.timerMotorLeg2On --;}
474 ////////////////////////////////////////////////////////////////////////////////
475         if(countMass.timerMotorBodyOn == 0 )
476         {
477             countMass.timerMotorBodyOn =
    TIME_MOTOR_1_2_ON_SWING;
478             countMass.timesMotorBodyOn++;
479             if(countMass.timesMotorBodyOn <
    TIMES_MOTOR_SWING )
480             {
481                 flag.settimerMotorBodyOn ^= 1;
482             }
483             else if(countMass.timesMotorBodyOn <
    TIMES_MOTOR_SWING+TIMES_MOTOR_NOT_SWING )
484             {
485                 flag.settimerMotorBodyOn = 1;
486             }
487             else
488             {
489                 countMass.timesMotorBodyOn = 0;
490             }
491         }
492         else if(countMass.timerMotorBodyOn > 0 ){
    countMass.timerMotorBodyOn --;}
493     }
494 ////////////////////////////////////////Timer 2
    ////////////////////////////////////////
495     if(TMR2IF == 1) // interval every 2.4 ms
496     {
```

```
497             TMR2IF = 0;
498             if(flag.EnMotorArm1)
499             {
500                 if(flag.Button1 == 1)
501                 {
502                     if(flag.settimerMotorArm1On == 1)
503                     {
504                         count_timePeriodMotor_Arm1 ++;
505                         if(count_timePeriodMotor_Arm1 >=
    TIME_PERIOD_PWM)
506                         {
507                             count_timePeriodMotor_Arm1 = 0;
508                             MOTOR5 = 1;
509                             MOTOR6 = 1;
510                         }
511                         else if(count_timePeriodMotor_Arm1 >=
    count.levelPWM_Arm1)
512                         {
513                             MOTOR5 = 0;
514                             MOTOR6 = 0;
515                         }
516                     //if(count_timePeriodMotor1 <
    TIME_PERIOD_PWM){count_timePeriodMotor1++;
    if(count_timePeriodMotor1 >= count.levelPWM_1){MOTOR_1 =
    OFF;}}
517                     }
518                     else MOTOR5 = 0; MOTOR6 = 0;
519                 }
520             }
521 ////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////
    //////
522         if(flag.EnMotorArm2)
523             {
524                 if(flag.Button2 == 1)
525                 {
526                     if(flag.settimerMotorArm2On == 1)
527                     {
528                         count_timePeriodMotor_Arm2 ++;
529                         if(count_timePeriodMotor_Arm2 >=
    TIME_PERIOD_PWM)
530                         {
531                             count_timePeriodMotor_Arm2 = 0;
532                             MOTOR7 = 1;
533                             MOTOR8 = 1;
534                         }
535                         else if(count_timePeriodMotor_Arm2 >=
    count.levelPWM_Arm2)
536                         {
537                             MOTOR7 = 0;
538                             MOTOR8 = 0;
539                         }
540                     //if(count_timePeriodMotor1 <
```

```
TIME_PERIOD_PWM){count_timePeriodMotor1++;
if(count_timePeriodMotor1 >= count.levelPWM_1){MOTOR_1 =
OFF;}}
541                 }
542                 else{ MOTOR7 = 0; MOTOR8 = 0;}
543             }
544         }
545 ////////////////////////////////////////////////////////////
////////////////////////////
546     if(flag.EnMotorLeg1)
547         {
548             if(flag.Button3 == 1)
549             {
550                 if(flag.settimerMotorLeg1On == 1)
551                 {
552                     count_timePeriodMotor_Leg1 ++;
553                     if(count_timePeriodMotor_Leg1 >=
TIME_PERIOD_PWM)
554                     {
555                         count_timePeriodMotor_Leg1 = 0;
556                         MOTOR9 = 1;
557                         MOTOR10 = 1;
558                     }
559                     else if(count_timePeriodMotor_Leg1 >=
count.levelPWM_Leg1)
560                     {
561                         MOTOR9 = 0;
562                         MOTOR10 = 0;
563                     }
564                 //if(count_timePeriodMotor1 <
TIME_PERIOD_PWM){count_timePeriodMotor1++;
if(count_timePeriodMotor1 >= count.levelPWM_1){MOTOR_1 =
OFF;}}
565                 }
566                 else{ MOTOR9 = 0; MOTOR10 = 0;}
567             }
568         }
569 ////////////////////////////////////////////////////////////
////////////////////////////
570     if(flag.EnMotorLeg2)
571         {
572             if(flag.Button4 == 1)
573             {
574                 if(flag.settimerMotorLeg2On == 1)
575                 {
576                     count_timePeriodMotor_Leg2 ++;
577                     if(count_timePeriodMotor_Leg2 >=
TIME_PERIOD_PWM)
578                     {
579                         count_timePeriodMotor_Leg2 = 0;
580                         MOTOR11 = 1;
581                         MOTOR12 = 1;
582                     }
```

```
583                     else if(count_timePeriodMotor_Leg2 >=
    count.levelPWM_Leg2)
584                     {
585                         MOTOR11 = 0;
586                         MOTOR12 = 0;
587                     }
588                 //if(count_timePeriodMotor1 <
    TIME_PERIOD_PWM){count_timePeriodMotor1++;
    if(count_timePeriodMotor1 >= count.levelPWM_1){MOTOR_1 =
    OFF;}}
589                 }
590                 else{ MOTOR11 = 0; MOTOR12 = 0;}
591             }
592         }
593 ////////////////////////////////////////////////////////////
    ////////////////////////
594         if(flag.EnMotorBody)
595         {
596             if(flag.Button5 == 1)
597             {
598                 if(flag.settimerMotorBodyOn == 1)
599                 {
600                     count_timePeriodMotor_Body ++;
601                     if(count_timePeriodMotor_Body >=
    TIME_PERIOD_PWM)
602                     {
603                         count_timePeriodMotor_Body = 0;
604                         MOTOR1 = 1;
605                         MOTOR2 = 1;
606                         MOTOR3 = 1;
607                         MOTOR4 = 1;
608                     }
609                     else if(count_timePeriodMotor_Body >=
    count.levelPWM_Body)
610                     {
611                         MOTOR1 = 0;
612                         MOTOR2 = 0;
613                         MOTOR3 = 0;
614                         MOTOR4 = 0;
615                     }
616                 //if(count_timePeriodMotor1 <
    TIME_PERIOD_PWM){count_timePeriodMotor1++;
    if(count_timePeriodMotor1 >= count.levelPWM_1){MOTOR_1 =
    OFF;}}
617                 }
618                 else{ MOTOR1 = 0;MOTOR2 = 0;MOTOR3 =
    0;MOTOR4 = 0;}
619             }
620         }
621     }
622 }
623 ////////////////////////////////////////////////////Timer 2
    ////////////////////////////////////////////
```

```
624
625  void main(void)
626   {
627      GIE = 1;         // Enable global interrupt
628      PEIE=1;
629      ANSEL = 0 ; // digital I/O
630      ANSELH = 0 ; // digital I/O
631      RBPU = 1 ; // 1 = PORTB pull-ups are disabled
632      Initial_timer1();
633      Initial_timer2();
634      Initial_Bluetooth();
635
636      Initial_Ports();
637      Initial_ADC();
638      __delay_ms(1000);
639      state = INITIAL_STATE;
640
641     while (1)
642      {
643          if(state == INITIAL_STATE)
644          {
645              countMass.timesMotorArm1On = 0;
646              countMass.timerMotorArm1On = 0;
647              countMass.timesMotorArm2On = 0;
648              countMass.timerMotorArm2On = 0;
649              countMass.timesMotorLeg1On = 0;
650              countMass.timerMotorLeg1On = 0;
651              countMass.timesMotorLeg2On = 0;
652              countMass.timerMotorLeg2On = 0;
653              countMass.timesMotorBodyOn = 0;
654              countMass.timerMotorBodyOn = 0;
655
656              flag.settimerMotorArm1On = 0;
657              flag.settimerMotorArm2On = 0;
658              flag.settimerMotorLeg1On = 0;
659              flag.settimerMotorLeg2On = 0;
660              flag.settimerMotorBodyOn = 0;
661
662              count.process_2_MArm1 = 1;
663              count.process_2_MArm2 = 1;
664              count.process_2_MLeg1 = 1;
665              count.process_2_MLeg2 = 1;
666              count.process_2_MBody = 1;
667
668              count.process_3_MArm1 = 2;
669              count.process_3_MArm2 = 2;
670              count.process_3_MLeg1 = 2;
671              count.process_3_MLeg2 = 2;
672              count.process_3_MBody = 2;
673
674
675              count.Arm1_low = 1;
676              count.Arm2_low = 2;
```

```
677             count.Leg1_low = 3;
678             count.Leg2_low = 4;
679             count.Back_low = 5;
680             count.OFF = 6;
681             count.Arm1_high = 7;
682             count.Arm2_high = 8;
683             count.Leg1_high = 9;
684             count.Leg2_high = 10;
685             count.Back_high = 11;
686
687             count_timeMs2_Arm1 = 0;
688             count_timeMs_Arm1 = 0;
689             count_timeSec2_Arm1 = 0;
690             count_timeSec_Arm1 = 0;
691
692             count_timeMs2_Arm2 = 0;
693             count_timeMs_Arm2 = 0;
694             count_timeSec2_Arm2 = 0;
695             count_timeSec_Arm2 = 0;
696
697             count_timeMs2_Leg1 = 0;
698             count_timeMs_Leg1 = 0;
699             count_timeSec2_Leg1 = 0;
700             count_timeSec_Leg1 = 0;
701
702             count_timeMs2_Leg2 = 0;
703             count_timeMs_Leg2 = 0;
704             count_timeSec2_Leg2 = 0;
705             count_timeSec_Leg2 = 0;
706
707             count_timeMs2_Body = 0;
708             count_timeMs_Body = 0;
709             count_timeSec2_Body = 0;
710             count_timeSec_Body = 0;
711
712             count_timePeriodMotor_Body = 0;
713             count_timePeriodMotor_Arm1 = 0;
714             count_timePeriodMotor_Arm2 = 0;
715             count_timePeriodMotor_Leg1 = 0;
716             count_timePeriodMotor_Leg2 = 0;
717
718             flag.M1 = 0;
719             flag.M2 = 0;
720             flag.M3 = 0;
721             flag.M4 = 0;
722             flag.M5 = 0;
723             flag.Button1 = 0;
724             flag.Button2 = 0;
725             flag.Button3 = 0;
726             flag.Button4 = 0;
727             flag.Button5 = 0;
728
729             flag.EnMotorArm1 = 0;
```

```
730             flag.EnMotorArm2 = 0;
731             flag.EnMotorLeg1 = 0;
732             flag.EnMotorLeg2 = 0;
733             flag.EnMotorBody = 0;
734
735             count.readKey = 0;
736
737             state = RUNNING_STATE;
738
739             flag.timeout = 0;
740             count.timeout = LIMIT_TIMEOUT;
741             count.timeoutMsec = 0;
742             countMass.timeout_Arm1 = LIMIT_TIMEOUT;
743             countMass.timeoutMsec_Arm1 = 0;
744             countMass.timeout_Arm2 = LIMIT_TIMEOUT;
745             countMass.timeoutMsec_Arm2 = 0;
746             countMass.timeout_Leg1 = LIMIT_TIMEOUT;
747             countMass.timeoutMsec_Leg1 = 0;
748             countMass.timeout_Leg2 = LIMIT_TIMEOUT;
749             countMass.timeoutMsec_Leg2 = 0;
750             countMass.timeout_Body = LIMIT_TIMEOUT;
751             countMass.timeoutMsec_Body = 0;
752
753
754         else if(state == RUNNING_STATE)
755
756
757             if(countMass.timeout_Arm1 == 0)
758             {
759                 flag.Button1 = 0;
760                 flag.M1 = 0;
761                 motor_Arm1(0);
762             }
763             if(countMass.timeout_Arm2 == 0)
764             {
765                 flag.Button2 = 0;
766                 flag.M2 = 0;
767                 motor_Arm2(0);
768             }
769             if(countMass.timeout_Leg1 == 0)
770             {
771                 flag.Button3 = 0;
772                 flag.M3 = 0;
773                 motor_Leg1(0);
774             }
775             if(countMass.timeout_Leg2 == 0)
776             {
777                 flag.Button4 = 0;
778                 flag.M4 = 0;
779                 motor_Leg2(0);
780             }
781             if(countMass.timeout_Body == 0)
782
```

```
783                     flag.Button5 = 0;
784                     flag.M5 = 0;
785                     motor_Body(0);
786                 }
787
788
789     ////////////////////////////////////////////////////////////
    ////////////
790                 else if(fn_read_key() == btnKEY1)
791                 {
792                      if(flag.Button1 == 0)
793                     {
794                         flag.Button1 = 1;
795                         flag.M1 = 1;
796                         countMass.timeoutMsec_Arm1 = 0;
797                         countMass.timeout_Arm1 = LIMIT_TIMEOUT;
798                         count_timePeriodMotor_Arm1 = 0;
799                         motor_Arm1(count.process_2_MArm1);
800                     }
801                     else
802                     {
803                         flag.Button1 = 0;
804                         flag.M1 = 0;
805                         motor_Arm1(0);
806                     }
807                     __delay_ms(TIME_DEBOUNCE);
808                     while(fn_read_key() == btnKEY1);
809                 }
810     ////////////////////////////////////////////////////////////
    ////////////
811                 else if(fn_read_key() == btnKEY2)
812                 {
813                      if(flag.Button2 == 0)
814                     {
815                         flag.Button2 = 1;
816                         flag.M2 = 1;
817                         countMass.timeoutMsec_Arm2 = 0;
818                         countMass.timeout_Arm2 = LIMIT_TIMEOUT;
819                         count_timePeriodMotor_Arm2 = 0;
820                         motor_Arm2(count.process_2_MArm2);
821                     }
822                     else
823                     {
824                         flag.Button2 = 0;
825                         flag.M2 = 0;
826                         motor_Arm2(0);
827                     }
828                     __delay_ms(TIME_DEBOUNCE);
829                     while(fn_read_key() == btnKEY2);
830                 }
831     ////////////////////////////////////////////////////////////
    ////////////
832                 else if(fn_read_key() == btnKEY3)
```

```
833                 {
834                     if(flag.Button3 == 0)
835                     {
836                         flag.Button3 = 1;
837                         flag.M3 = 1;
838                         countMass.timeoutMsec_Leg1 = 0;
839                         countMass.timeout_Leg1 = LIMIT_TIMEOUT;
840                         count_timePeriodMotor_Leg1 = 0;
841                         motor_Leg1(count.process_2_MLeg1);
842                     }
843                     else
844                     {
845                         flag.Button3 = 0;
846                         flag.M3 = 0;
847                         motor_Leg1(0);
848                     }
849                     __delay_ms(TIME_DEBOUNCE);
850                     while(fn_read_key() == btnKEY3);
851                 }
852 ////////////////////////////////////////////////////////////////////
    ///////////
853                 else if(fn_read_key() == btnKEY4)
854                 {
855                     if(flag.Button4 == 0)
856                     {
857                         flag.Button4 = 1;
858                         flag.M4 = 1;
859                         countMass.timeoutMsec_Leg2 = 0;
860                         countMass.timeout_Leg2 = LIMIT_TIMEOUT;
861                         count_timePeriodMotor_Leg2 = 0;
862                         motor_Leg2(count.process_2_MLeg2);
863                     }
864                     else
865                     {
866                         flag.Button4 = 0;
867                         flag.M4 = 0;
868                         motor_Leg2(0);
869                     }
870                     __delay_ms(TIME_DEBOUNCE);
871                     while(fn_read_key() == btnKEY4);
872                 }
873 ////////////////////////////////////////////////////////////////////
    ///////////
874                 else if(fn_read_key() == btnKEY5)
875                 {
876                     if(flag.Button5 == 0)
877                     {
878                         flag.Button5 = 1;
879                         flag.M5 = 1;
880                         countMass.timeoutMsec_Body = 0;
881                         countMass.timeout_Body = LIMIT_TIMEOUT;
882                         count_timePeriodMotor_Body = 0;
883                         motor_Body(count.process_2_MBody);
```

```
884                 }
885                 else
886                 {
887                     flag.Button5 = 0;
888                     flag.M5 = 0;
889                     motor_Body(0);
890                 }
891                 __delay_ms(TIME_DEBOUNCE);
892                 while(fn_read_key() == btnKEY5);
893             }
894 ////////////////////////////////////////////////////////////////////////////
895             else if(fn_read_key() == btnKEY6)
896             {
897                     All_OFF();
898 
899                 __delay_ms(TIME_DEBOUNCE);
900                 while(fn_read_key() == btnKEY6);
901             }
902             read_BT = BT_getc();
903             switch(read_BT)
904             {
905                 case 'A':
906                 BT_Process(count.Arm1_low);
907                 break;
908 
909                 case 'B':
910                 BT_Process(count.Arm1_high);
911                 break;
912 
913                 case 'C':
914                 BT_Process(count.Arm2_low);
915                 break;
916 
917                 case 'D':
918                 BT_Process(count.Arm2_high);
919                 break;
920 
921                 case 'E':
922                 BT_Process(count.Leg1_low);
923                 break;
924 
925                 case 'F':
926                 BT_Process(count.Leg1_high);
927                 break;
928 
929                 case 'G':
930                 BT_Process(count.Leg2_low);
931                 break;
932                 case 'H':
933                 BT_Process(count.Leg2_high);
934                 break;
935 
```

```
936                 case 'I':
937                 BT_Process(count.Back_low);
938                 break;
939
940                 case 'J':
941                 BT_Process(count.Back_high);
942                 break;
943
944                 case 'K':
945                 BT_Process(count.OFF);
946                 break;
947
948                 default : read_BT = 0;
949             }
950 ////////////////////////////////////////////////////////////////////
    ///////////
951         }
952     }
953  }
954 void Initial_Bluetooth(void)
955 {
956     //Set the buad rate
957     BRGH = 1;
958     SPBRG = 25;
959     // Turn on Async Serial Port
960     SYNC = 0;
961     SPEN = 1;
962
963     //Enable Tx,Rx
964     CREN = 1;
965     TXEN = 1;
966
967 }
968 void BT_putc(unsigned char c)
969 {
970     while(!TRMT);
971     TXREG = c;
972 }
973 void BT_puts(unsigned char *s)
974 {
975     while(*s)
976     {
977         BT_putc(*s++);
978     }
979 }
980
981 unsigned char BT_getc()
982 {
983     if(RCIF)
984     {
985         while(!RCIF);
986         return(RCREG);
987     }
```

```
988   }
989   void BT_Process(unsigned char process)
990   {
991               if(process == 1)
992               {
993                    if(flag.Button1 == 0)
994                   {
995                       flag.Button1 = 1;
996                       flag.M1 = 1;
997                       countMass.timeoutMsec_Arm1 = 0;
998                       countMass.timeout_Arm1 = LIMIT_TIMEOUT;
999                       count_timePeriodMotor_Arm1 = 0;
1000                      motor_Arm1(count.process_2_MArm1);
1001                  }
1002                  else
1003                  {
1004                      flag.Button1 = 0;
1005                      flag.M1 = 0;
1006                      motor_Arm1(0);
1007                  }
1008              }
1009  ////////////////////////////////////////////////////////////
      ///////////
1010              else if(process == 2)
1011              {
1012                   if(flag.Button2 == 0)
1013
1014                      flag.Button2 = 1;
1015                      flag.M2 = 1;
1016                      countMass.timeoutMsec_Arm2 = 0;
1017                      countMass.timeout_Arm2 = LIMIT_TIMEOUT;
1018                      count_timePeriodMotor_Arm2 = 0;
1019                      motor_Arm2(count.process_2_MArm2);
1020
1021                  else
1022
1023                      flag.Button2 = 0;
1024                      flag.M2 = 0;
1025                      motor_Arm2(0);
1026                  }
1027              }
1028  ////////////////////////////////////////////////////////////
      ///////////
1029              else if(process == 3)
1030              {
1031                   if(flag.Button3 == 0)
1032                  {
1033                      flag.Button3 = 1;
1034                      flag.M3 = 1;
1035                      countMass.timeoutMsec_Leg1 = 0;
1036                      countMass.timeout_Leg1 = LIMIT_TIMEOUT;
1037                      count_timePeriodMotor_Leg1 = 0;
1038                      motor_Leg1(count.process_2_MLeg1);
```

```
1039                }
1040                else
1041                {
1042                    flag.Button3 = 0;
1043                    flag.M3 = 0;
1044                    motor_Leg1(0);
1045                }
1046            }
1047  ////////////////////////////////////////////////////////////
    ///////////
1048            else if(process == 4)
1049            {
1050                if(flag.Button4 == 0)
1051                {
1052                    flag.Button4 = 1;
1053                    flag.M4 = 1;
1054                    countMass.timeoutMsec_Leg2 = 0;
1055                    countMass.timeout_Leg2 = LIMIT_TIMEOUT;
1056                    count_timePeriodMotor_Leg2 = 0;
1057                    motor_Leg2(count.process_2_MLeg2);
1058
1059                }
1060                else
1061                {
1062                    flag.Button4 = 0;
1063                    flag.M4 = 0;
1064                    motor_Leg2(0);
1065
1066                }
1067  ////////////////////////////////////////////////////////////
    ///////////
1068            else if(process == 5)
1069            {
1070                if(flag.Button5 == 0)
1071                {
1072                    flag.Button5 = 1;
1073                    flag.M5 = 1;
1074                    countMass.timeoutMsec_Body = 0;
1075                    countMass.timeout_Body = LIMIT_TIMEOUT;
1076                    count_timePeriodMotor_Body = 0;
1077                    motor_Body(count.process_2_MBody);
1078                }
1079                else
1080                {
1081                    flag.Button5 = 0;
1082                    flag.M5 = 0;
1083                    motor_Body(0);
1084                }
1085            }
1086  ////////////////////////////////////////////////////////////
    ///////////
1087            else if(process == 6)
1088                All_OFF();
```

```
1089                }
1090    ////////////////////////////////////////////////////////////
    ////////////
1091                if(process == 7)
1092                {
1093                     if(flag.Button1 == 0)
1094                    {
1095                        flag.Button1 = 1;
1096                        flag.M1 = 1;
1097                        countMass.timeoutMsec_Arm1 = 0;
1098                        countMass.timeout_Arm1 = LIMIT_TIMEOUT;
1099                        count_timePeriodMotor_Arm1 = 0;
1100                        motor_Arm1(count.process_3_MArm1);
1101                    }
1102                    else
1103                    {
1104                        flag.Button1 = 0;
1105                        flag.M1 = 0;
1106                        motor_Arm1(0);
1107                    }
1108
1109    ////////////////////////////////////////////////////////////
    ////////////
1110                else if(process == 8)
1111                {
1112                     if(flag.Button2 == 0)
1113                    {
1114                        flag.Button2 = 1;
1115                        flag.M2 = 1;
1116                        countMass.timeoutMsec_Arm2 = 0;
1117                        countMass.timeout_Arm2 = LIMIT_TIMEOUT;
1118                        count_timePeriodMotor_Arm2 = 0;
1119                        motor_Arm2(count.process_3_MArm2);
1120                    }
1121                    else
1122                    {
1123                        flag.Button2 = 0;
1124                        flag.M2 = 0;
1125                        motor_Arm2(0);
1126                    }
1127                }
1128    ////////////////////////////////////////////////////////////
    ////////////
1129                else if(process == 9)
1130                {
1131                     if(flag.Button3 == 0)
1132                    {
1133                        flag.Button3 = 1;
1134                        flag.M3 = 1;
1135                        countMass.timeoutMsec_Leg1 = 0;
1136                        countMass.timeout_Leg1 = LIMIT_TIMEOUT;
1137                        count_timePeriodMotor_Leg1 = 0;
1138                        motor_Leg1(count.process_3_MLeg1);
```

```
1139                }
1140                else
1141                {
1142                    flag.Button3 = 0;
1143                    flag.M3 = 0;
1144                    motor_Leg1(0);
1145                }
1146            }
1147 ////////////////////////////////////////////////////////////
     ///////////
1148             else if(process == 10)
1149            {
1150                if(flag.Button4 == 0)
1151                {
1152                    flag.Button4 = 1;
1153                    flag.M4 = 1;
1154                    countMass.timeoutMsec_Leg2 = 0;
1155                    countMass.timeout_Leg2 = LIMIT_TIMEOUT;
1156                    count_timePeriodMotor_Leg2 = 0;
1157                    motor_Leg2(count.process_3_MLeg2);
1158
1159                else
1160                {
1161                    flag.Button4 = 0;
1162                    flag.M4 = 0;
1163                    motor_Leg2(0);
1164
1165
1166 ////////////////////////////////////////////////////////////
     ///////////
1167             else if(process == 11)
1168             {
1169                 if(flag.Button5 == 0)
1170
1171                 {
1172                     flag.Button5 = 1;
1173                     flag.M5 = 1;
1174                     countMass.timeoutMsec_Body = 0;
1175                     countMass.timeout_Body = LIMIT_TIMEOUT;
1176                     count_timePeriodMotor_Body = 0;
1177                     motor_Body(count.process_3_MBody);
1178                 }
1179                 else
1180                 {
1181                     flag.Button5 = 0;
1182                     flag.M5 = 0;
1183                     motor_Body(0);
1184                 }
1185             }
1186
1187 }
1188 void Initial_timer1(void)
1189 {
        T1CKPS1 = 1;
```

```
1190        T1CKPS0 = 1;
1191        //Set Prescale 1:8
1192        TMR1CS = 0; //Use internal clock
1193        T1SYNC = 0; // Synchronize external clock input
1194        TMR1ON = 1; //Enable Timer1
1195        TMR1IF = 0; //The timer1 register did not overflow
1196        TMR1IE = 1; //Enable Overflow Interrupt bits
1197        TMR1H = 0x0B;
1198        TMR1L = 0xDB; // 0.5 sec//65535 - 56075 = 9460
1199
1200        //At 4 MHz
1201        //Frequency : Fosc/4/8 = 4MHz/4/8 = 125kHz
1202        //clock time period= 1/Freq : 0.000008 = 8 us
1203        //16 bits = 65535 bits
1204        //max time = 8 us x 65535 = 0.52428 sec
1205        // 1 ms : timer count = required time / clock time period

1206        // 1 ms/ 0.000008 = 125 counts
1207    }
1208    void Initial_timer2(void)
1209    {
1210        T2CKPS1 = 1;
1211        T2CKPS0 = 0;
1212        //Set Prescale 1:16
1213        TMR2ON = 1; //Enable Timer2
1214        TMR2IF = 0; //A TMR2 to PR2 match occurred.
1215        TMR2IE = 1; //TMR2 to PR2 Match interrupt Enabled.
1216        TMR2 = 0;
1217        PR2 = 150; //max 8bit or 0xff //lengh of time peroid pwm
1218
1219        //At 4 MHz
1220        //Frequency : Fosc/4/8 = 4MHz/4/16 = 62.5kHz
1221        //clock time period= 1/Freq : = 16 us
1222        //8 bits = 255 bits
1223        //max time = 16 us x 150 = 0.00248 sec = 2.4 ms.
1224        //F = 1/0.004 = 250 Hz.
1225        // 0.004/(1/(4000000/4/16)) = 250
1226
1227    }
1228
1229    unsigned int analog(unsigned char channel)
1230    {
1231        unsigned int value = 0;
1232        if(channel >= 0 && channel <= 13)
1233        {
1234            //CHS<3:0>: Analog Channel Select bits
1235            CHS3 = (channel & 0x08) == 0x08 ;
1236            CHS2 = (channel & 0x04) == 0x04 ;
1237            CHS1 = (channel & 0x02) == 0x02 ;
1238            CHS0 = (channel & 0x01) == 0x01 ;
1239
1240            GODONE = 1;  // Start conversion
1241            while(GODONE);  // Wait until conversion success
```

```
1242            value = (ADRESH<<2)+(ADRESL>>6);      // Get ADC
    value
1243        }
1244        return value ;
1245    }
1246    void Initial_ADC(void)
1247    {
1248        /*ADFM = 1;
1249        VCFG0 = 0; //VDD as Voltage Ref.
1250        VCFG1 = 0;
1251        ADON = 1; // A/D Enabled.
1252        ADCS1 = 1;// F RC (clock derived from a dedicated
    internal oscillator = 500 kHz max)
1253        ADCS0 = 1;
1254        // config port that you want
1255        ANS0 = 1;
1256    */
1257        ADCON0 = 0b11010001;
1258        //ADCON1 = 0b10000000;
1259        ANS4 = 1;
1260    }
1261    void motor_Arm1 (unsigned char level)
1262    {
1263        if(level == 0){ flag.EnMotorArm1 = 0;  MOTOR5 = 0;MOTOR6
    = 0;}
1264        else if(level == 1){ flag.EnMotorArm1 = 1;
    count.levelPWM_Arm1 = PWM_LEVEL_1;}
1265        else if(level == 2){ flag.EnMotorArm1 = 1;
    count.levelPWM_Arm1 = PWM_LEVEL_2;}
1266        else if(level == 3){ flag.EnMotorArm1 = 1;
    count.levelPWM_Arm1 = PWM_LEVEL_3;}
1267        else if(level == 4){ flag.EnMotorArm1 = 1;
    count.levelPWM_Arm1 = PWM_LEVEL_4;
1268        countMass.timesMotorArm1On = 0;
1269        countMass.timerMotorArm1On = 0;
1270        flag.settimerMotorArm1On = 0;
1271    }
1272    void motor_Arm2 (unsigned char level)
1273    {
1274        if(level == 0){ flag.EnMotorArm2 = 0;  MOTOR7 = 0;MOTOR8
    = 0;}
1275        else if(level == 1){ flag.EnMotorArm2 = 1;
    count.levelPWM_Arm2 = PWM_LEVEL_1;}
1276        else if(level == 2){ flag.EnMotorArm2 = 1;
    count.levelPWM_Arm2 = PWM_LEVEL_2;}
1277        else if(level == 3){ flag.EnMotorArm2 = 1;
    count.levelPWM_Arm2 = PWM_LEVEL_3;}
1278        else if(level == 4){ flag.EnMotorArm2 = 1;
    count.levelPWM_Arm2 = PWM_LEVEL_4;}
1279        countMass.timesMotorArm2On = 0;
1280        countMass.timerMotorArm2On = 0;
1281        flag.settimerMotorArm2On = 0;
1282    }
```

```
1283  void motor_Leg1 (unsigned char level)
1284  {
1285      if(level == 0){ flag.EnMotorLeg1 = 0;  MOTOR9 =
      0;MOTOR10 = 0;}
1286      else if(level == 1){ flag.EnMotorLeg1 = 1;
      count.levelPWM_Leg1 = PWM_LEVEL_1;}
1287      else if(level == 2){ flag.EnMotorLeg1 = 1;
      count.levelPWM_Leg1 = PWM_LEVEL_2;}
1288      else if(level == 3){ flag.EnMotorLeg1 = 1;
      count.levelPWM_Leg1 = PWM_LEVEL_3;}
1289      else if(level == 4){ flag.EnMotorLeg1 = 1;
      count.levelPWM_Leg1 = PWM_LEVEL_4;}
1290      countMass.timesMotorLeg1On = 0;
1291      countMass.timerMotorLeg1On = 0;
1292      flag.settimerMotorLeg1On = 0;
1293  }
1294  void motor_Leg2 (unsigned char level)
1295  {
1296      if(level == 0){ flag.EnMotorLeg2 = 0;  MOTOR11 =
      0;MOTOR12 = 0;}
1297      else if(level == 1){ flag.EnMotorLeg2 = 1;
      count.levelPWM_Leg2 = PWM_LEVEL_1;}
1298      else if(level == 2){ flag.EnMotorLeg2 = 1;
      count.levelPWM_Leg2 = PWM_LEVEL_2;}
1299      else if(level == 3){ flag.EnMotorLeg2 = 1;
      count.levelPWM_Leg2 = PWM_LEVEL_3;}
1300      else if(level == 4){ flag.EnMotorLeg2 = 1;
      count.levelPWM_Leg2 = PWM_LEVEL_4;}
1301      countMass.timesMotorLeg2On = 0;
1302      countMass.timerMotorLeg2On = 0;
1303      flag.settimerMotorLeg2On = 0;
1304  }
1305  void motor_Body (unsigned char level)
1306  {
1307      if(level == 0){ flag.EnMotorBody = 0;  MOTOR1 = 0;MOTOR2
      = 0;MOTOR3 = 0;MOTOR4 = 0;}
1308      else if(level == 1){ flag.EnMotorBody = 1;
      count.levelPWM_Body = PWM_LEVEL_1;}
1309      else if(level == 2){ flag.EnMotorBody = 1;
      count.levelPWM_Body = PWM_LEVEL_2;}
1310      else if(level == 3){ flag.EnMotorBody = 1;
      count.levelPWM_Body = PWM_LEVEL_3;}
1311      else if(level == 4){ flag.EnMotorBody = 1;
      count.levelPWM_Body = PWM_LEVEL_4;}
1312      countMass.timesMotorBodyOn = 0;
1313      countMass.timerMotorBodyOn = 0;
1314      flag.settimerMotorBodyOn = 0;
1315  }
1316  int fn_read_key(void)
1317  {
1318      read_adc = analog(4); //read PIN AN0
1319      __delay_ms(2);
1320      if(read_adc < REF_KEY_1){ return btnKEY1;}
```

```
1321        if(read_adc < REF_KEY_2){ return btnKEY2;}
1322        if(read_adc < REF_KEY_3){ return btnKEY3;}
1323        if(read_adc < REF_KEY_4){ return btnKEY4;}
1324        if(read_adc < REF_KEY_5){ return btnKEY5;}
1325        if(read_adc < REF_KEY_6){ return btnKEY6;}
1326        return btnNONE;
1327    }
1328    int fn_read_BT(void)
1329    {
1330            read_BT = BT_getc();
1331            __delay_ms(2);
1332            if(read_BT == 'A'){return btnKEY1;}
1333            if(read_BT == 'B'){return btnKEY2;}
1334            if(read_BT == 'C'){return btnKEY3;}
1335            if(read_BT == 'D'){return btnKEY4;}
1336            if(read_BT == 'E'){return btnKEY5;}
1337            if(read_BT == 'F'){return btnKEY6;}
1338            return btnNONE;
1339    }
1340
1341    void All_OFF(void)
1342    {
1343        flag.Button1 = 0;
1344        flag.Button2 = 0;
1345        flag.Button3 = 0;
1346        flag.Button4 = 0;
1347        flag.Button5 = 0;
1348        flag.M1 = 0;
1349        flag.M2 = 0;
1350        flag.M3 = 0;
1351        flag.M4 = 0;
1352        flag.M5 = 0;
1353        motor_Arm1(0);
1354        motor_Arm2(0);
1355        motor_Leg1(0);
1356        motor_Leg2(0);
1357        motor_Body(0);
1358    }
1359
1360    void Initial_Ports(void)
1361    {
1362        ANSEL  = 0;
1363        ANSELH  = 0;
1364        C1ON = 0; // C1 Comparator is disabled
1365        C2ON = 0; // C2 Comparator is disabled
1366
1367        MOTOR1_TRIS = 0;//output
1368        MOTOR2_TRIS = 0;//output
1369        MOTOR3_TRIS = 0;//output
1370        MOTOR4_TRIS = 0;//output
1371        MOTOR5_TRIS = 0;//output
1372        MOTOR6_TRIS = 0;//output
1373        MOTOR7_TRIS = 0;//output
```

```
1374        MOTOR8_TRIS = 0;//output
1375        MOTOR9_TRIS = 0;//output
1376        MOTOR10_TRIS = 0;//output
1377        MOTOR11_TRIS = 0;//output
1378        MOTOR12_TRIS = 0;//output
1379
1380        MOTOR1 = 0;// LOW
1381        MOTOR2 = 0;// LOW
1382        MOTOR3 = 0;// LOW
1383        MOTOR4 = 0;// LOW
1384        MOTOR5 = 0;// LOW
1385        MOTOR6 = 0;// LOW
1386        MOTOR7 = 0;// LOW
1387        MOTOR8 = 0;// LOW
1388        MOTOR9 = 0;// LOW
1389        MOTOR10 = 0;// LOW
1390        MOTOR11 = 0;// LOW
1391        MOTOR12 = 0;// LOW
1392    }
1393
```